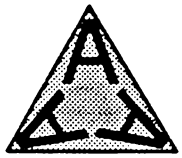
An Overview of the Advanced Amiga Architecture and Other Future Directions



Document Revison 1.0

1993 Developer's Conference Release

by Dave Haynie
Advanced Amiga System Group

Copyright ©1993 Commodore International Services Corp, Technology Group
All Rights Reserved

Information contained herein is the unpublished, confidential and trade secret property of Commodore International Services Corporation, Technology Group. Use, reproduction, or disclosure of this information without the prior explicit written permission of Commodore is strictly probibited.

Chapter 1
Introduction

Since the introduction of the original Amiga 1000 in 1985, there have been two major revisions of the Amiga chip set: ECS and AA. The ECS chip set introduced a minimally upgraded version of Agnus and Denise, while the AA chip set introduced more significant enhancements, including Lisa, a brand new Denise replacement. Despite the advantages wrought by AA, the AA chip set was still very much an evolution of the original Amiga chip set rather than anything revolutionary. AA maintained most of the original features of Agnus and the entire Paula chip, chainging only those things related to video display.

The AAA chip set is the first Amiga chip set to break from this original Amiga architecture. It is composed of four completely new full custom VLSI integrated circuits. It improves every aspect of the Amiga chip set's performance, and its new architecture makes possible many things that could never be directly adapted to the original architecture in any practical sense.

1.1 Targets

This paper is intended to be an overview of the AAA chip set and the direction we're going in with respect to Systems architecture that will surround the AAA chips. This is not intended as a complete definition of either of these, we have hundreds of pages of internal documentation devoted to AAA and next generation systems architecture that does that job.

A good understanding of the original, ECS, and AA chip sets, while probably not vital, will be very helpful in understanding this document and, more importantly, the goals of AAA itself. While there have been many, many improvements in AAA over previous Amiga chip sets, AAA is still very much an Amiga chip set. It is an updated version of many of the same design philosophies that led to prior Amiga implementations.

1.2 Credits

Most of the material on the AAA chip set was gleaned, adapted, and sometimes outright copied from the internal document "Advanced AMIGA Architecture". Many thanks go out to Jim Redfield, Ed Hepler, and the rest of the AAA design group for writing most of this paper for me.

-

Chapter 2
Goals of the AAA Project

The main point of the AAA chip set is to move the Amiga back toward the leading edge of personal computer technology. However, in doing so, one main requirement is basic compatibility with the existing Amiga chip sets, to keep as much software running as possible without compromising the other AAA goals.

2.1 Compatibility

AAA is designed to be largely register compatible with the ECS chip set. Most of the RGA registers from ECS are supported. The ECS "Ultra hires" registers have been eliminated, as they were never supported in actual practice. Some other display-generation details of ECS are no longer required or supported in AAA.

The AA registers are not supported in AAA. We believe that the 3.0 OS provides the necessary control over AA and that no one need program "to the metal" on AA systems. Additionally, some of the AA features were implemented in a less-than-ideal fashion, in order to fit in the same RGA address space originally implemented in ECS. All AA-equivalent function can be done much better by new AAA support than some kind of AA emulation. Some behaviors can't be perfectly emulated in AAA. Clearly, the AAA chip set's architecture will have an immediate impact on some elements of the ECS emulation apart from register-level compatibility. For example, on a VRAM system, there's no way to slow the system down to an equivalent cycle-stealing ECS mode. So a program will often find significantly more blitter, copper, and CPU access than on an ECS machine. This won't be a problem if you're using the OS correctly, but it could be a problem to take-over-the-system type programs. Such programs may also have some degree of problem with some AAA screen promotions and copper activity.

We envision AAA as a transitional system. It is highly desirable to eliminate the ECS-compatibility, or for that matter, reliance on any register-level compatibility, for the next generation of Amiga chips. So there's an excellent chance that you're seeing some of this stuff for the last time in AAA.

2.2 Flexibility

The pre-ECS, ECS, and to a lesser degree the AA chip sets were each designed with a single system architecture in mind. This defined a good deal of how the chips work together, what kind and how much memory they would support, which CPU interface they'd hook into, etc. Any deviation from this basis could result in a pricey system full of work-around logic, as we had to some extent in the A3000. They also forced this defined Amiga chip sub-system to be the same in all computers, from the low-end to the high-end, so we might have too much expense for an ideal low end, not enough power for an ideal high-end.

The AAA chips are designed to work in several configurations. They can use cheap DRAM, but will gain significant extra performance using VRAM (DRAM with a high speed serial port). They can easily hook up to a variety of 32-bit CPU buses via an easy-to-implement asynchronous slave interface. CPU access to Chip RAM can be gated though similarly to ECS or AA, or the CPU bus can master the entire chip bus (providing all RAM timing) for more efficient CPU to Chip bus access. Finally, the AAA chips can be assembled in "Single" or "Dual" configurations, depending on the display goals.

2.3 Improvements

Every aspect of the previous Amiga chip sets has been improved upon in AAA. While not every feature can be discussed here at length, some of the improvements follow.

2.3.1 Memory Bandwidth

The AAA chip bus is an improvement over the ECS and even AA chip bus in terms of memory speed. The AAA chips run a four cycle burst to Chip RAM, which in raw performance is 4.56 times faster than ECS memory access or 1.14 time faster than AA's two-cycle burst. However, the real key to AAA's memory architecture is its support for VRAM. With VRAM, display fetches have practically no effect on the normal parallel chip RAM bus, freeing it for use by Blitter, Copper, and CPU.

Page-Mode performance is actually a bit better than this implies, since under the right circumstances the AAA chips can run extended burst cycles. Bursts of up to 512 words can be run to keep up with high resolution displays. This improves overall system performance, but increases latency to chip RAM.

2.3.2 CPU Bandwidth

The CPU access to Chip RAM is improved over past systems. Since the AAA chips manage an asynchronous interface to the CPU, CPUs running at any speed take less of a synchronization penalty for chip RAM access than they do in the current A3000 and A4000 systems, where synch-up is managed externally by the Gary chip. Also, AAA's dynamic chip bus slot allocation allows the CPU to get in more often, it's not limited to one out of every two slots. Finally, as mentoned, an external device such as the CPU with some extra support logic can completely master the AAA chip RAM bus, allowing Chip RAM access as fast as today's 32-bit Fast RAM.

2.3.3 Chip Bus DMA

Unlike previous Amiga systems, the AAA chip bus DMA activity is dynamically managed. The different DMA channels (40 of them, including the standard and high-priority external channels for CPU) have fixed priorities with respect to one another, though the relation between the blitter and the CPU can be adjusted in various ways. Because of this dynamic allocation, its possible to run out of cycles before everything has all the time it wants. The

highest priority channels are the deterministic channels, the lowest the blitter or CPU. They are: graphics overlay, DRAM refresh, interrupt transfer, disk, high priority external access, audio, display, sprite, coprocessor, processor, and blitter (the latter two by default and changable, as mentioned).

When too many cycles are requested, something requiring deterministic acccess has to starve. That something will be the graphics fetch. When the system can't fetch enough graphics data for a line, it sends the graphics overrun interrupt to the CPU. This is to be taken as an indicator to software than something needs to be quieted down.

2.3.4 Blitter

The AAA blitter is significantly faster than the ECS/AA blitter when running in 32-bit mode, thanks to the faster, wider bus. Just doing basic scrolls, it can scroll a 640x200x2 screen about 6 times faster, or a 640x200x4 screen about 9 times faster than with the old blitter. But that's just raw data movement.

Logical improvments to the blitter streamline much of its use. In 32-bit mode, it's much easier to program. It now operates using pixel addressing rather than via masks, modulos, and shifts. It can operate on traditional Amiga bitplanes, or on chunky pixels of 2, 4, 8, or 16 bits width. The line-draw has also been improved, supporting a new "clip-rect" mode for better GUI performance under Intuition. Finally, several arithmetic operations have been added for "sort" and "tally" operations on planes of any pixel depth.

2.3.5 Copper

The copper has been improved in several areas. It can handle 32-bit operations for the new 32-bit registers, and supports a "move-multiple" function for more efficient loading of blocks of consecutive registers, such as color tables. The copper now has an interrupt capability, which lets it receive an interrupt from the blitter. This allows the copper to manage a series of blit operations, one after another, without additional processor intervention.

2.3.6 Graphics

Extensive improvements have been made to the graphics in AAA. A "single" system with Fast-Page DRAM can support displays up to 800x560x9 bitplanes. The same system using VRAM can support 800x560x13 bitplanes or 800x560x24 using "hybrid" pixels. A "dual" system can support up to 1280x1024x5 displays with Fast-Page DRAM, up to 1280x1024x8 bitplanes or 1024x768x24 "hybrid" using VRAM.

The AAA chip set supports 256 CLUT entries of 25 bits each, like AA does. It can handle sprites up to 128 bits wide. It supports up to 16 bitplanes, which makes dual 8-bit playfields possible. The AAA pixel clock is no longer ties to the AAA bus clock, so a variety of display resolutions, even standard ones, can be generated by an AAA system. Hardware-assisted screen promotion is also supported via scaled pixel clocks.

There are a large variety of pixel types supported in AAA. Along with the traditional bitplane-generated pixels (including HAM8 and HAM10), we have several kinds of chunky and compressed pixels.

Half-Chunky	Half-chunky pixels come in 2, 4, or 8-bit depths. These indirect through the color lookup table like most planar modes do.
Chunky	Chunky pixels are 16-bits deep. They bypass the CLUT, providing 5 bits of red, 5-bits of green, 5-bits of blue, and one genlock overlay directly.
Hybrid	Hybrid pixels are 24-bits deep, composed of separate chunky planes for Red, Green, and Blue. The genlock overlay is fixed at {R,G,B} = 0 for this mode.
PACKLUT	These compressed pixels are stored at 2 bits per pixel and decompress to 8-bit half-chunky pixels. This is done by dividing the screen into 4 x 4 pixel regions. Each region contains two colors (8-bit values indexed through the CLUT) and sixteen pixels.
PACKHY	These compressed pixels are stored at 4 bits per pixel and decompress to 24-bit direct pixels. This is done by dividing the screen into 4 x 4 pixel regions. Each region contains two colors (24-bit direct values) and sixteen pixels.

2.3.7 Video Capture

The AAA pixel bus direction can be reversed, allowing an optional low cost video capture device (framegrabber) to be implemented in AAA systems. Capture can be in any chunky display mode. This only works in systems that use VRAM.

2.3.8 Sound

The AAA chip set contains the first improvement in Amiga-based sound since the Amiga was introduced. The audio circuitry can handle sampling rates of better than 50kHz with 16-bit resolution. Eight channels are supported, and channels can be assigned to the left or right output. The 16-bit D/A converters are on-chip, and an external converter is also supported. Additionally, the chip set does 8-bit audio sampling.

2.3.9 Floppy Disk

In addition to supporting the original 1 megabyte disk used in previous Amiga systems, the AAA chip set supports 2 and 4 megabyte disk formats as well. This is, of course, direct support, no speed controls or other kludges are necessary.

As well as supporting the increased floppy densities, the AAA chipset has considerably more flexibility. It has built-in decoding hardware, which can decode MFM, RLL(2,7), and Biphase Mark (CD-ROM) formats. It can transfer by track, sector, a special "CD mode", and a special high speed track mode. Its data I/O rate has increased from 0.5 Mbit/sec to approximately 11.4 Mbit/sec (though only encoded data can handle this, since the DMA rate available to the floppy logic peaks at 9.9Mbits/sec). Finally, the data separator is programmable, which is very useful at tweaking up to the optimal performance with any specific medium.

This flexible controller can handle practically any floppy format yet invented. With the proper software, it should have no problem with the original Mac format. It handles IBM formats at 360K, 720K, 1.2MB, 1.44MB, and 2.88MB, directly with sectoring (no track buffer necessary). In theory, an RLL(2,7) floppy format at 4 megabyte density could store somewhere between 4.0MB and 5.2MB on a single disk. It should also be able to support 21.6MB flopticals.

And it can support devices other than floppies, too. CD-ROM would be a direct connect, and similarly formatted DAT and digital radio should work too. It might even be possible to support an ST-506 hard drive (assuming such puppies still exist). Higher transfer rates need a cleaner signal, and may require external clocking of the data separator PLL, which is supported by the chip set.

2.3.10 UARTS

The AAA chip set contains two UARTs. Both are improvements over the Paula UART, each buffered with a four-byte FIFO. This significantly improves serial performance at high speeds, since fewer interrupts are taken, and increases reliability, since there's much more time available to respond to a serial interrupt.

Chapter 3
The AAA Chips

The AAA system consists of four completely new VLSI chips, implemented in high speed CMOS. The functions of the first three are partitioned similarly to those of the ECS/AA chip sets. Andrea is the chip bus controller, analogous to Agnus/Alice. Monica is the new display controller, replacing Denise. Mary, the Paula replacement, controls various types of I/O. Finally, a new chip, Linda, double-buffers full display lines.

For the most part, the four chips function as one. Between them, there are nearly 256 word-addressed registers (compatibility registers), 384 longword-addressed registers (new stuff), and 512 longword-addressed CLUT registers. It's often true that a single register address function is performed by two or three of the chips.

3.1 The Andrea Chip

Andrea is the chip bus controller, responsible for managing the chip bus. It is the core of the AAA system, much like a microprocessor is the core of a normal CPU system. The Andrea chip has a rather impressive list of features:

- Chip RAM control. Normally, Andrea manages all chip RAM access, supporting both Fast-Page DRAM and VRAM. A simple configuration mechanism tells Andrea what's on the chip bus at startup time. Eight 256Kword banks of chip RAM are supported, and with a little extra logic banks of Fast-Page DRAM and VRAM can be intermixed on a bank-by-bank basis. A "high priority" bus request allows an external device to master the chip RAM bus rather than Andrea.

- CPU bus gating. The Andrea chip controls access to the chip RAM bus and chip registers from the CPU port. CPU addresses go through Andrea, CPU data is externally latched under Andrea's control. This is a reasonably general purpose CPU interface, which manages synchronization to the chip bus, directly supporting CPUs of varying clock speeds (previously this was done with extra logic wrapped around the ECS and AA chip sets, like the A3000 and A4000).

  Chip bus control. A variety of control signals for management of both the chip bus and various aspects of the other AAA chips are generated in Andrea. The multiplexed chip address/data bus is also mastered by Andrea. This bus contains a register address at the start of a cycle (like the RGA bus on ECS/AA), data in the later stanges of a cycle. All of the AAA chip set's DMA channels are addressed by Andrea, which is also responsible for allocating the various DMA channels according to which units are requesting DMA and the priority of each requesting channel.

- Clocks. The Andrea chip manages clocking of both the chip bus and the video display. Control lines from Andrea select one of eight possible pixel clock values available at any given time. These can be changed on a line-by-line basis.

Video timing control. All video timing and synchronization signals are created in Andrea. A number of different counters in Andrea generate video synchs, blanking, and the logical controls necessary to support the AAA display. Andrea also manages a light pen intput.

Blitter. The Andrea chip contains the Amiga blitter. This blitter can handle the AAA chip RAM bus at full speed and width, but it also has a compatibility mode that lets it handle 16-bit data just like the pre-AAA blitter. In 32-bit bit mode, the blitter has a considerable number of new functions. The new blitter is pixel addressed, it automatically calculates first and last mask, proper shifts, and whether pre-reads are necessary. It supports pixel sizes of 1, 2, 4, 8, or 16 bits, to cover all AAA display modes. It now has "sort" and "tally" functions designed mainly to assist chunky pixel processing. The sort function will run a bubble sort on a plane of pixels, governed by a sort key. The tally function records the number of instances of each byte value in a plane of pixel data. Finally, the new blitter can perform a variety of arithmetic operations on chunky pixels, including addition, averaging, subtraction, saturated subtraction, etc. The following tables illustrate example blitter speed (screens scrolled/second) for various system configuration:

Single, Fast-Page DRAM

Display	640 x 200	640 x 400	800 x 560	1024 x 768	1280 x 1024
2 Bitplane	489.06	233.42	124.54	NA	NA
4 Bitplane	233.37	105.58	51.49	NA	NA
8 Bitplane	105.53	41.65	14.96	NA	NA
16 Bitplane	41.61	NA	NA	NA	NA
Half Chunky	110.22	46.33	19.03	NA	NA
Chunky	46.52	14.59	NA	NA	NA
Hybrid	24.99	NA	NA	NA	NA
ECS 2 Bitplane	81.80	25.83	NA	NA	NA
ECS 4 Bitplane	25.87	NA	NA	NA	NA

Single, Video DRAM

Display	640 x 200	640 x 400	800 x 560	1024 x 768	1280 x 1024
2 Bitplane	504.23	233.42	124.54	NA	NA
4 Bitplane	248.54	105.58	51.49	NA	NA
8 Bitplane	120.70	41.65	14.96	NA	NA
16 Bitplane	56.78	NA	NA	NA	NA
Half Chunky	127.39	46.33	19.03	NA	NA
Chunky	63.70	14.59	NA	NA	NA
Hybrid	42.17	NA	NA	NA	NA
ECS 2 Bitplane	81.80	25.93	NA	NA	NA
ECS 4 Bitplane	25.87	NA	NA	NA	NA

Dual, Fast-Page DRAM

Display	640 x 200	640 x 400	800 x 560	1024 x 768	1280 x 1024
2 Bitplane	498.24	242.59	134.05	71.66	38.56
4 Bitplane	242.56	114.75	61.00	30.05	13.59
8 Bitplane	114.72	50.83	24.48	9.24	NA
16 Bitplane	50.80	18.87	NA	NA	NA
Half Chunky	118.77	54.86	27.72	11.93	3.44
Chunky	55.06	23.12	9.63	3.31	NA
Hybrid	33.54	NA	NA	NA	NA
ECS 2 Bitplane	81.80	25.83	NA	NA	NA
ECS 4 Bitplane	25.87	NA	NA	NA	NA

Dual, Video DRAM

Display	640 x 200	640 x 400	800 x 560	1024 x 768	1280 x 1024
2 Bitplane	504.36	248.69	140.49	78.67	46.38
4 Bitplane	248.68	120.85	67.44	37.06	21.41
8 Bitplane	120.84	56.93	30.91	16.25	8.92
16 Bitplane	56.92	24.97	12.65	5.85	2.68
Half Chunky	126.85	62.93	35.71	20.12	11.92
Chunky	63.36	31.40	17.79	9.99	5.89
Hybrid	41.62	20.32	11.36	6.25	NA
ECS 2 Bitplane	81.80	25.83	NA	NA	NA
ECS 4 Bitplane	25.87	NA	NA	NA	NA

- Copper. The AAA copper, as mentioned, supports both 16-bit and 32-bit register operations. A multiple move instruction has been implemented to greatly reduce the overhead of large sequential register movements. A new interrupt mechanism allows the copper to be interrupted. Interrupt sources, in order of priority, are vertical blanking, wait finished, and blitter finished. The blitter finished interrupt allows the copper to re-load the blitter with the next blit operation at the end of the current blitter operation. In this way, the CPU can usually just schedule blit operations in the copper's blitter interrupt routine and get on with other work.

3.2 The Linda Chip

The Linda chip is a display line buffer for the AAA system. Linda provides a great deal of the intelligence behind the AAA display system. While one complete line of display data is being fed to Monica from one of Linda's line buffers, the next line is being fetched into Linda's other line buffer. There are many advantages of this prefetch process:

- Much more efficient Fast-Page display fetch cycles can be run than previously possible. Previous systems use zero or two-cycle Fast-Page fetches, AAA systems can run "burst" cycles hundreds of transfers in length.

- Bitplane alignment can still be on word boundries (as allowed with ECS), Linda makes any necessary realignments in order to pass longword-aligned data on to Monica. There are, however, various alignment restrictions on some of the new display modes:

Mode	Alignment
Bitplane	16-bit
Half-Chunky	64-bit
Chunky	64-bit
PACKLUT	64-bit
PACKHY	64-bit
Overlay	128-bit
32-bit Sprites	16-bit
64-bit Sprites	64-bit
128-bit Sprites	128-bit

  Rather than Fast-Page memory, Video RAM (VRAM) can be used without requiring the strict alignment requirements typical of most VRAM-based display systems. VRAM allows display fetch overhead to be essentially eliminated from normal chip bus activity.

- The pixel clock is easily decoupled from the bus clock. The chip RAM side of Linda runs at chip RAM bus speed, the Monica side of Linda runs at pixel clock speed.

- Packed pixel formats PACKLUT and PACKHY are decoded here, passing them on to Monica as normal 8-bit half-chunky or 24-bit hybrid pixels, respectively.

3.3 The Monica Chip

Monica is the AAA display controller chip. It takes in display timing data generated by Andrea and graphics data fetched by Linda and from that generates 25-bit digital and analog RBGoutput (24-bits of color, one of genlock overlay). Monica contains the 256 entry CLUT (color lookup table), HAM mode logic, priority control for playfields/overlay and sprite display, and 8-bit digital to analog converters, one each for Red, Green, and Blue.

HAM (hold and modify) mode is of course the special compact high-color display mode provided in ECS and AA chip sets. Using this mode, Monica can either supply a direct CLUT value or modify a previously displayed value. The five and six-bit modes from ECS, with 4096 color resolution, is supported, as well as the eight-bit mode from ECS/AA and a new ten-bit HAM, which allows a full 24-bits worth of color resolution using only ten bitplanes.

An optional one-bit overlay plane is also supported in Monica for chunky or packed display modes. This requires a VRAM-based chip RAM buffer and the chunky or packed display must be in the odd playfield, while the overlay is fetched based on the even playfield pointer. There are a number of restrictions on the overlay plane when compared to a normal second playfield in the traditional bitplane display.

The actual display capability of any given system is ultimately determined by how fast Monica can be fed pixel data by Linda. Which is, of course, determined by how fast Linda can fetch data from the chip RAM bus. This depends on the type of memory, the size of the system (single or dual), and the display mode (chunky modes tend to allow a more efficient fetch from chip RAM for the same resolution). The following tables show non-interlaced resolutions with different system setups.

Single, Fast-Page DRAM

Display Resolution	Bitplane	Half Chunky	Chunky	Hybrid	Pack LUT	Pack HY
640 x 200	16	yes	yes	yes	yes	yes
704 x 200	16	yes	yes	yes	yes	yes
640 x 400	12	yes	yes	no	yes	yes
800 x 560	10	yes	no	no	yes	no
1024 x 768	NA	NA	NA	NA	NA	NA
1280 x 1024	NA	NA	NA	NA	NA	NA

Single, Video DRAM

Display Resolution	Bitplane	Half Chunky	Chunky	Hybrid	Pack LUT	Pack HY
640 x 200	16	yes	yes	yes	yes	yes
704 x 200	16	yes	yes	yes	yes	yes
640 x 400	16	yes	yes	yes	yes	yes
800 x 560	14	yes	yes	yes	yes	yes
1024 x 768	NA	NA	NA	NA	NA	NA
1280 x 1024	NA	NA	NA	NA	NA	NA

Dual, Fast-Page DRAM

Display Resolution	Bitplane	Half Chunky	Chunky	Hybrid	Pack LUT	Pack HY
640 x 200	16	yes	yes	yes	yes	yes
704 x 200	16	yes	yes	yes	yes	yes
640 x 400	16	yes	yes	yes	yes	yes
800 x 560	13	yes	yes	yes	yes	yes
1024 x 768	8	yes	yes	no	yes	yes
1280 x 1024	5	yes	no	no	yes	no

Dual, Video DRAM

Display Resolution	Bitplane	Half Chunky	Chunky	Hybrid	Pack LUT	Pack HY
640 x 200	16	yes	yes	yes	yes	yes
704 x 200	16	yes	yes	yes	yes	yes
640 x 400	16	yes	yes	yes	yes	yes
800 x 560	16	yes	yes	yes	yes	yes
1024 x 768	11	yes	yes	yes	yes	yes
1280 x 1024	8	yes	yes	no	yes	yes

3.4 The Mary Chip

Mary is the AAA peripheral controller chip. It manages floppy disk, audio, and serial (UART) I/O. It is the first improvement in these areas for Amigas built into an Amiga chip set, and the improvements are considerable.

The floppy disk system was discussed in some detail in Chapter 2. Basically, the data separator now runs up to twenty times faster, with variable parameters and the option of a separate PLL clock. Mary supports internal decode of several formats, several new transfer options, hardware CRC calculation, and a higher DMA bandwidth to chip RAM. A comparison with Paula details these new features:

Function	Paula	Mary
raw bit width	2000, 4000 ns	88-9000 ns
max raw bit rate	0.5 Mbit/sec	11.4 Mbit/sec
encoding methods:		
raw	yes	yes
GCR	yes	yes
MFM	no (done in software)	yes
RLL(2,7)	no	yes
Biphase-Mark	no	yes
sync	16-bit	32, 16, 8-bit
transfer modes:		
track	yes	yes
sector	no	yes
CD digital	no	yes
"trackplus"	no	yes
hardware CRC	no	yes (sector, trackplus)
async PLL clock	no	yes
input types	pulse	pulse, NRZ

Mary's audio system is also a substantial improvement over Paula's. The sampling rate,

number of channels, sample resolution, volume resolution, sampling accuracy, and flexibility have all been improved. On-chip DACs are now at 16-bit resolution, and a digital audio output is available at 20-bits per channel (the supported samples are 16-bits, but sample volume yields a 17-bit result, and eight channels summed into a single output without scaling yields a 20-bit

Function	Paula	Mary
sample rate	29kHz	64kHz
channels	4	8
volume bits	6	12 (signed)
volume aliasing	yes	no
sample size	8 bits	8 or 16 bits
digital out	no	yes
dynamic range	15 bits	16 bits
channels on left	2 (0,3)	0-8 (any or all)
channels on right	2 (1,2)	0-8 (any or all)
global mono bit	no	yes
ouput time resolution	280 ns	280 ns
period resolution	280 ns	280/64 ns

value). A comparison of audio in Paula versus Mary yields:

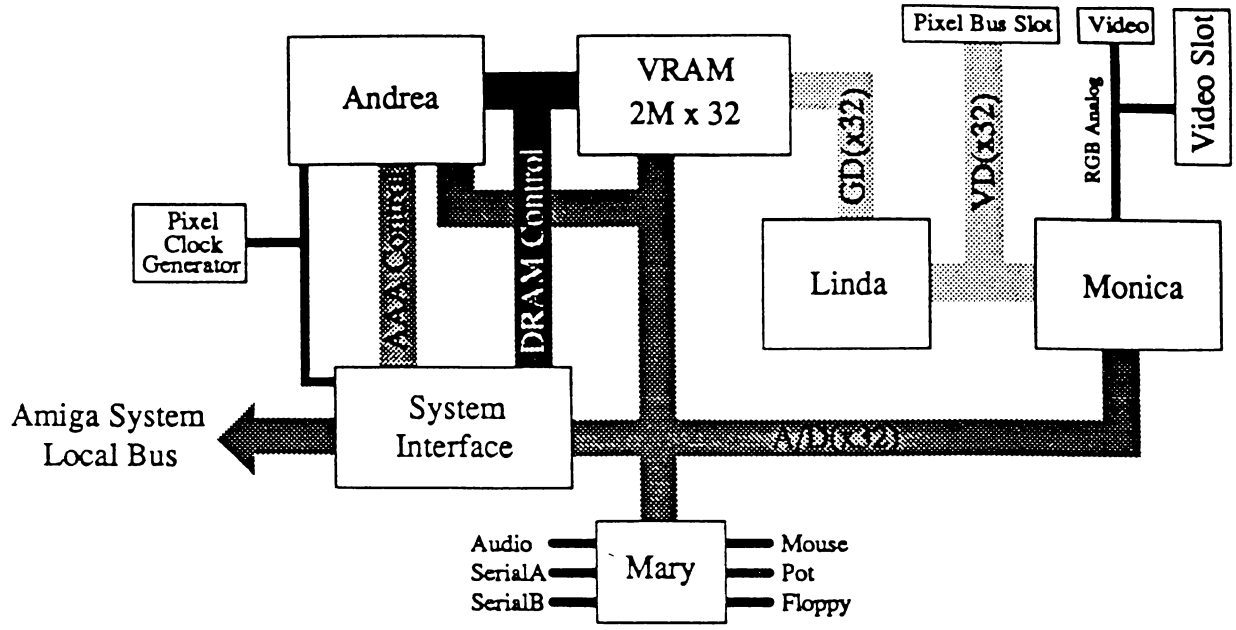
The pot inputs in Mary have been enhanced over Paula's in separate ways. First of all, in order to make them consistent between the widely varying display frequencies supported in AAA, Mary maintains its own sampling counter, which is basically an NTSC compatible horizontal line counter. There is also a new audio sampling mode, which turns the pot lines around to support 8-bit stereo input (with appropriate analog interface circuitry). Higher resolutions are available via an external ADC.

As mentioned before, the Mary UART is an enhancement over the Paula UART. A four deep FIFO has been added, which will reduce interrupt overhead (and overruns caused by missed interrupts) and make faster serial reads possible. Mary contains two UARTS, one that's in the same RGA address as the original Paula UART.

3.5 The Single System

The "single" AAA subsystem consists of one each of the four AAA chips. This example system uses video RAM, which is certainly the preferred implementation. Andrea controls this VRAM and the chip bus. The 32-bit parallel port of the VRAM connects to Andrea and the A/D bus (register address and all data go across this bus). The serial ports of the VRAM connect to Linda, which handles all the video fetching, under Andrea's direction. Mary connects to the A/D bus and to various I/O ports, such as audio, floppy, and serial. Finally, Monica connects to the other side of Linda, the A/D bus, and to the video output (digital or analog).

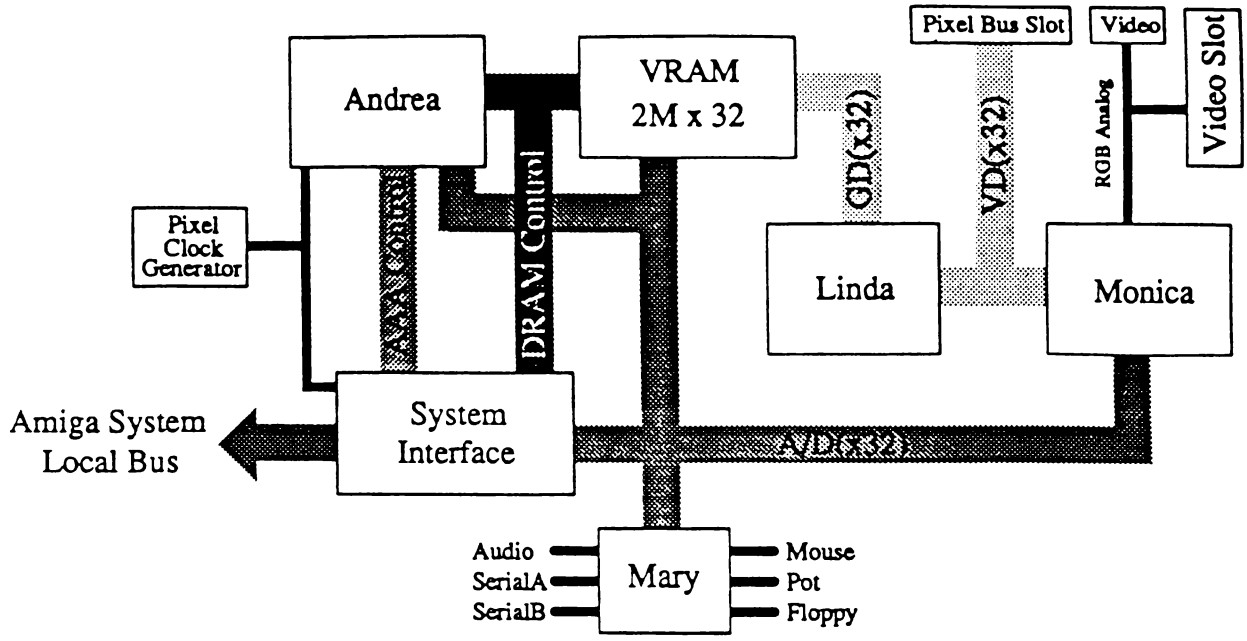
The interface to any kind of system will be implemented in some kind of "glue" chip, most likely a new gate array, though this function certainly could be implemented in a PAL and



TTL based circuit, as on the AAA prototype system. Ideally, uch a glue chip will do a variety of jobs. It should buffer data between the CPU/Local bus and the chip bus. A four longword deep buffer here will allow a CPU to write to chip RAM without an immediate delay, and it can also be used to match data rates between the chip and local buses, which are most likely running at different speeds. Andrea requires some kind of external logic to properly drive RAS* lines to each DRAM bank -- this also might be implemented in such a glue device. This will be especially true if its desirable to build an AAA system that handles VRAM in some kind of SIMM, since a SIMM wouldn't naturally drop into AAA's addressing and AUTOCONFIG scheme. The glue chip may also have something to do with pixel clock generation, though the main pixel clock generator will more than likely be a custom clock synthesis device.

The "pixel bus slot" shown is an expansion slot that makes the pixel bus (video data bus) available for expansion. This isn't the pixel output, but the raw data sent from memory to Monica. The main reason for providing some kind of expansion here is to support the framegrabber mode of AAA. In this mode, Linda reverses direction. Data from the pixel bus (supplied by a video capture bus of some kind, in an appropriate format) goes into Linda and is read out and stored by Andrea. This mode also requires the system to be slaved to an external pixel clock, like in genlock mode.

Single AAA systems can also support a traditional AA-compatible video slot. This can support devices that use either analog video or 25-bit digital video. Not every AA-compatible video device will necessarily work here, of course, since the pixel clock and display rates can be significantly higher than those possible in AA systems. It's not obvious, however, that a new type of video slot would be a better solution, though perhaps some of the reserved pins on the AA slot specification will be used for extra AAA signals in actual AAA implementations.



3.6 The Dual System

An alternate AAA system configuration is the so-called "dual" system. This system uses an Andrea and Mary as before, but contains two Lindas and two Monicas. Andrea, Mary, and Monica still sit on a 32-bit random-access bus, but the display path is now 64-bits wide. Lindas and Monicas are paired in "even" and "odd" groups to process, in turn, even and odd pixels. The pixel rate generated at output is twice the pixel clock rate, so pixel rates as high as 110 MHz can be reached with this kind of system.

In order to support HAM mode, which is of course dependent on past pixels, each Monica indicates its last CLUT access and HAM operation on a special HAM bus, and in turn snoops the HAM bus of the other Monica. In order to generate the full-speed video output and still use the Monica video DACs, each Monica chip has a direction control for its pixel bus. In this setup, the even Monica puts pixel information out, while the odd Monica takes that same information in and feeds it, along with its own pixel data, to the DACs. Both 24-bit values go to the DACs in the same pixel clock.

This system supports a 64-bit pixel bus for video capture and other similar operations. It's possible to support an AA style 25-bit digital video slot as well, but with even more limitations. Since the pixel bus of the odd Monica acts as input, only every other pixel is actually available in digital form on this dual system. In some resolutions, that won't be a problem, in others it will. A full featured digital video port would require an external DAC and either a multiplexer or 48-bit pixel path. A better solution for this kind of thing would be to adopt some kind of digital video transmission protocol as soon as one becomes available.

Chapter 4
Future System Concerns

The AAA chips obviously function as just part of a whole Amiga system. It's certainly possible to build a machine, like the Amiga 500, where the Amiga chips essentially are the whole system. Such a system would be composed of the AAA chips, a microprocessor, memory, some CIAs, analog stuff for audio, and one gate array for "glue". Of course, such a system is rather boring to talk about. It's also somewhat unlikely that early AAA systems will be of this flavor.

Taking the high-end route, there's considerably more to an Amiga system than the custom chips. It would, however, be a mistake to base things directly on past high-end systems like the A3000/A4000. The system architecture of those systems, while fine for the time, is lacking in a number of important areas. The most obvious factor is that the A3000 architecture isn't very modular or flexible. It was designed to support our ideas for the A3000, nothing more. Even in the A4000, extra logic was necessary to push this A3000 architecture is a slightly different direction. On the AAA prototype motherboard, there is extensive high-speed PAL logic necessary to implement a servicable but unsophisticated AAA system.

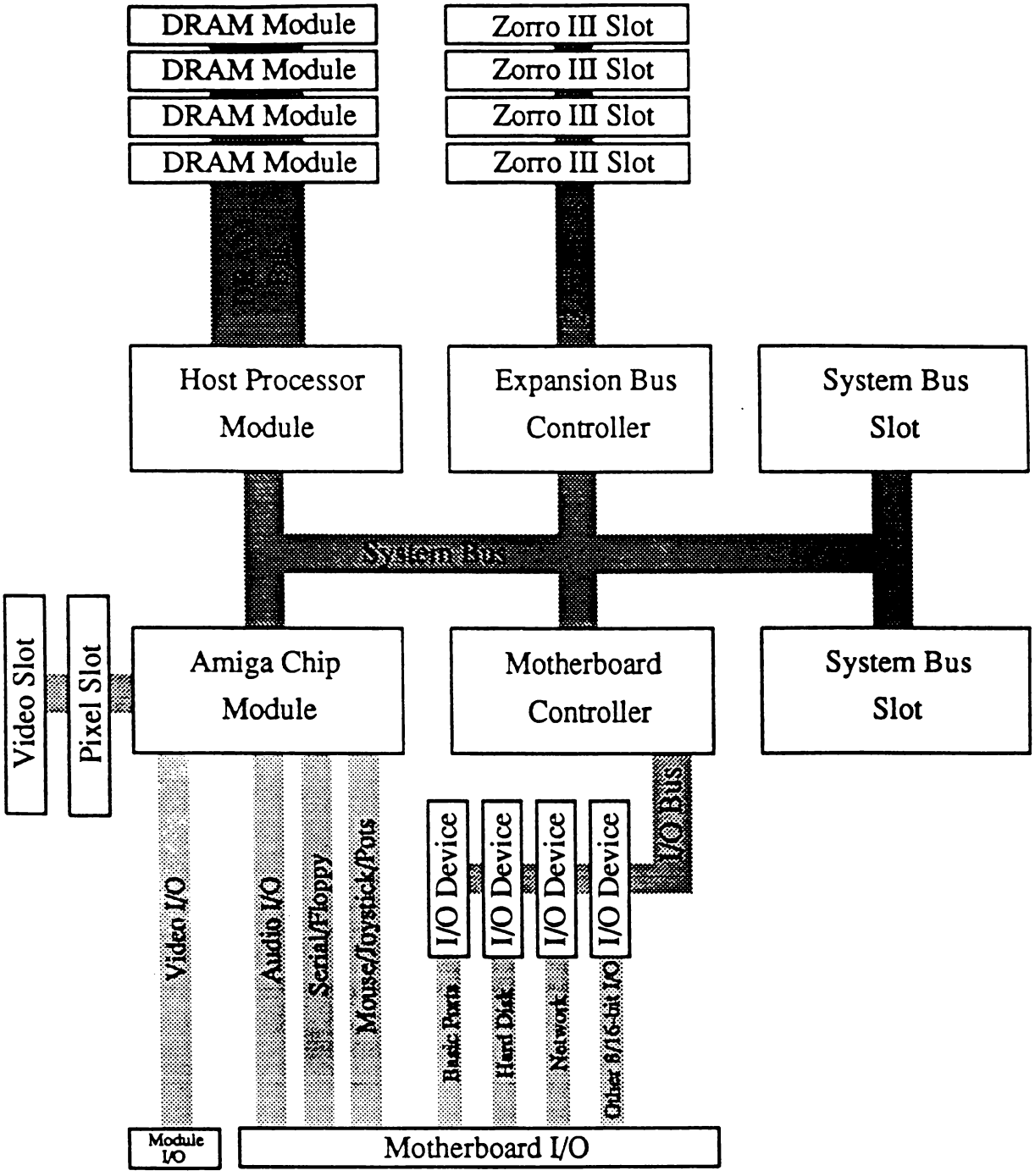
The primary goal of an advanced Amiga system can be summed up in one word: *modulatity*. Such a new system, both logically and physically, is composed of several interchangable subsystems. No one piece has any unnatural dependence on any other; interconnections between the system components have to be based on intentional system standards, not chance implementation details.

4.1 The System Bus

The next generation system should have a processor-independent system bus optimised for chip to chip interconnect. For the most part this replaces the traditional CPU-specific local bus found on previous Amiga systems. This establishes a standard to which several generations of new system and, eventually, Amiga chips can be designed. Since each major system chip hooks into the system bus independently of any other, this finally breaks the interdependence of chips in a chip set, allowing upgrades as necessary to any piece of the system.

4.2 The Motherboard

The motherboard for such a system contains just the basics that will be needed by every system. This will certainly include a number of basic I/O chips for the standard ports on that machine. The CPU, Amiga chips, and various other elements of the system are located on separate modules. These don't necessarily have to be physically located on different cards, but ideally they will be. Not only does this make motherboard upgrade much easier, but it allows several different motherboards to be designed using the same plug-in modules, and it allows Commodore to easily support more options in system and processor makeup.



The heart of the motherboard is the motherboard controller. This manages motherboard based I/O, such as CIA chips, hard disk interface (IDE, SCSI, whatever), network, etc. This also acts as a support for the system bus, handling arbitration of bus master and interrupts. Most I/O chips sit on a flexible 8/16 bit I/O bus defined by the system controller. Many programmable chip selects and interrupt inputs allow new I/O devices to be added as time goes by without extra glue or the need to redesign the motherboard controller.

4.3 The Amiga Chip Module

For the first time in an Amiga system, the Amiga chips are located on a plug-in module rather than fixed into the motherboard. This certainly allows easy upgrade to new Amiga chip sets, but its also a very reasonably thing to consider just in light of the AAA chip set, since a veriety of configurations can be made just with AAA, even if no other Amiga chips were considered. The Amiga module connects to the system bus and to a special motherboard-support connector. The system interface device glues Andrea to the system bus, providing the proper bus translation, data FIFOing, and chip selects. The support connector routes general Amiga resources, such as audio, floppy and UART lines, to the appropriate I/O connectors on the motherboard. Digital and analog video also go off this way to mate with video expansion connectors. The main video connector and other connectors specific to the Amiga module will be provided directly on the Amiga module.

4.4 The Host Processor Module

The main system processor is supported on the host processor module. This card has one connector to the system bus, and it gets the first system bus access on boot up by default. It has a second connector to a wide DRAM bus located on the motherboard. While the motherboard houses this DRAM, it's totally up to the host module to drive the DRAM bus. This of course means that the host module must contain a DRAM controller. In most cases, the DRAM controller will be intregrated with the system bus controller to provide a low cost interface between host CPU and the motherboard. Locating the DRAM controller on the host module allows DRAM, the most speed-critical element of the system, to be optimized for any host processor chosen.

4.5 The Expansion Controller

The expansion controller is another system bus chip that does conversions to and from the Zorro bus protocols. This can be located on a motherboard (maybe for towers) or on a system bus module (maybe for desktops). It is completely optional -- if a Zorro bus isn't desired, or should be an option on some systems, fine.

4.6 Open System Bus Slots

There will likely be a free system bus slot or two on most expandable machines. These support various kinds of high-speed expansion: fast peripherals, processor farms, DSPs, etc. The number of open system bus slots will of course depend on the final system bus specification, the space available on a given motherboard, etc. Due to the anticipated speed and purpose of such slots, it's not expected that there will be more than two or three open slots in any system, and the module cards aren't likely to be very large. These aren't designed to replace Zorro III as a general purpose expansion bus. Instead, the system bus should be thought of as a more flexible local bus/coprocessor slot.

Retargetable Graphics Specification

Copyright ©1992,1993 Commodore Electronics Ltd.

Proposed RTG system requirements:

ECS or greater.
Separate ROMS for ECS/AA and AAA via conditional compilation/assembly.
'020 or greater required, at least for initial release.

A Graphics Driver

A graphics device driver is better viewed as a set of graphics routines being used by the graphics library, rather than as a library in its own right. Core differences are:

- The vector table size is determined at install time by graphics.library. This is done in order to allow the future extension by inclusion of new primitives in terms of older primitives, and to allow graphics to install its preferred default entry points.

- The device driver pointer is passed to the entry point in A5. GfxBase is kept in A6 for the execution of the function. Register conventions follow the normal standard (a0/a1 d0/d1 are scratch, return values in d0) unless otherwise specified.

Driver Functions

This section defines the functions needed to initialize a driver.

```
APTR CreateDriverA(struct TagList *tags)
```

creates a driver for a display device, and opens it. It returns either a pointer to the device base, or NULL if an error occurred.

TAGS:

DRV_InitVectorTable Points to a list specifying which default vectors should be replaced with driver- specific ones. (fmt= UWORD nentries, UWORD firstentry, ULONG &code)

DRV_CloneVectorTable Points to another graphics driver. Initial contents of the vector table are taken from this driver, instead of defaults.

The Graphics Database

This section describes the graphics database, and the role it plays in the Retargetable Graphics model.

Outline

The Amiga graphics system is capable of displaying many different types of display modes. As all these display modes have varying characteristics, such as colors, dimensions, etc, a database exists describing all the characteristics of all the modes; this removes the need for application writers to make assumptions about the display modes, especially as the characteristics can vary depending on machine configuration, chip type etc.

Each mode has a unique identifier ModeID, and is used as a key to the database functions.

The MonitorSpec

Although the MonitorSpec structure is publicly defined, it is highly unlikely that many developers will be looking at it. The only elements of the MonitorSpec that might be of use are the total_rows and total_colorclocks, which are used to calculate the monitor's refresh rates. The remainder of the structure is very specific to the Amiga Chip Set (in fact, so is the total_colorclocks, but we may have to live with that). However, there are still some elements that all the MonitorSpecs should have as a minimum.

Because of the fact that the MonitorSpec is mainly of importance to only the graphics library, it seems reasonable to redefine the structure as follows. I have tried to keep as many offsets in the same place as before as possible.

```
struct  MonitorSpec
{
    struct      ExtendedNode ms_Node;
    UWORD       ms_Flags;               /* driver specific flags */
    LONG        MonitorID;
    APTR        DriverBase;             /* points to the driver base */
    UWORD       total_rows;             /* # of lines per field - same offset */
    UWORD       total_colorclocks;      /* line time in 280ns units */
    ULONG       DisplayCompatible;      /* how this mode coexists */
    UWORD       min_row;                /* at the same offset as before */
    APTR        GfxPrivate;             /* ours! (It's at the same place as a
                                         * struct SpecialMonitor * is now).
                                         */
    UWORD       OpenCount;              /* At the same offset as before */
    struct      List ModeIDs;           /* Database entries hang off here */
    UWORD       MoreFlags;
    struct      Rectangle ms_LegalView; /* at the same offset as before*/
    struct      MonitorInfo *minfo;     /* Pointer to a MonitorInfo structure
                                         * that is inherited by all the ModeIDs
                                         * of this monitor. */
    UBYTE       DriverData[n];          /* driver specific data */
}
```

The MonitorSpec is now expandable, as 'n' is undefined. In the case of the native Amiga drivers, the DriverData contains much the same data as the V37 MonitorSpec.

The MonitorID is used to identify the monitor spec. For example, graphics would need to know if this is a MonitorSpec for the Amiga driver. In that case, it would know that it could look at the ms_Flags field.

New DataBase Functions

The bit layout of a monitor ID is as follows:

```
1nnn nnnn nnss ssuu
```

Bit 15 identifies a third party monitor type.

"nnnnnnnn" is the device code, which will be registered with us. "ssss" is the scan-rate subcode, which selects (for instance) different scan rates for the device. "uu" is the unit number of the device. The unit number is automagically incremented by CreateMonitorA (see below) when multiple units are added.

```
struct MonitorSpec *CreateMonitorA(struct NewMonitor *nm, struct TagList *tags)
```

creates a new MonitorSpec, and adds it to the list of monitors. It will return a pointer to a MonitorSpec, or NULL if error. This function will create a MonitorSpec large enough for the defined part plus the number of bytes in the driver specific area, and copy the driver specific data into the MonitorSpec.

```
struct NewMonitor
{
        ULONG   nm_MonitorID;    /* Pointer to requested Monitor number. On
                                  * successful return, this field will be
                                  * set to the actual monitor number given.
                                  */
        STRPTR  nm_MonitorName;  /* Pointer to a string of the base monitor
                                  * name. The string will be copied to the
                                  * MonitorSpec.
                                  */
        APTR    nm_DriverBase;   /* Pointer to the DriverBase for this monitor */
        ULONG   nm_DeviceNumber;
                                 /* A number describing what physical device
                                  * views with this monitor will appear on. 0
                                  * Means the normal Amiga video output.
                                  * This should be gotten with GetUniqueID().
                                  */
        UWORD   nm_PreferredModeID;
                                 /* For preferences. The lower word of the whole
                                  * ID that Overscan.prefs displays the editor
                                  * screen in.
                                  */
        UWORD   nm_TotalRows;    /* Total Rows */
        UWORD   nm_TotalCClocks;
                                 /* Total ColorClocks per line
                                  * (1 ColorClock = 280ns)
                                  */
        UWORD   nm_Pad;
        struct Rectangle nm_LegalView;
                                 /* The LegalView rectangle */
        struct Point nm_ViewResolution;
                                 /* Monitor ticks-per-pixel */
};
```

TAGS:

AM_TextRange - pointer to a struct Rect32, for the TextRange overscan setting. Defaults to the NominalRange in the DimensionInfo of the PreferredModeID.

AM_GfxRange - Pointer to a struct Rect32 for the StdRange overscan setting. Defaults to TxtOScan.

AM_MS_Flags - Driver Specific flags. Defaults to 0

AM_MinRow - Defaults to 0.

AM_DataSize - The number of bytes in the DriverData

AM_DriverData - Points to driver specific data (requires the AM_DataSize tag).

AM_Compat - Compatibility - default is MCOMPAT_SELF

AM_DefaultViewPos - (for MonitorInfo) - defaults to (0, minrow)

AM_ViewPosition - (for MonitorInfo) - defaults to DefaultViewPos

Each call to CreateMonitorA() will cause the driver to be opened. That way it is possible to attach more than one monitor to a driver. For example, the AA driver will have all the Amiga monitors attached to it.

[NB - currently, the MonitorInfo has a pad[36] where the RawMonitorInfo has a Rect32 TxtOScan and a Rect32 StdOScan. We need to define these in displayinfo.h so the driver can add these too.]

If the monitor number is not the first monitor with that number (more than one identical board in the system), CreateMonitorA() will add a '1', '2' etc to the name. So, if this is the second "NTSC" monitor in the system (being on a plug-in AA card, with the native ECS chips being the first", CreateMonitorA() will make the name "NTSC1" for this monitor.

```
DisplayInfoHandle AddModeIDA(ULONG ModeID, struct TagList *tags)
```

This will create a new ModeID entry if this doesn't already exist, or add the specified data to an existing entry. If the specified data already exists for the ModeID, AddModeIDA() will return INVALID_ID as the handle. It is up to the driver to initialize the various entries properly. This is done either with subsequent calls to AddModeID() to initialize the entries for the first time, or SetDisplayInfoData() to modify entries.

TAGS:

AMID_Dims - Data points to an initialized struct DimensionInfo
AMID_Disp - Data points to an initialized struct DisplayInfo
AMID_Name - Data points to a string to name the ModeID
AMID_Driver - Data points to the DriverBase
AMID_DriverData - Data points to mode/driver specific data.

AMID_Dims, AMID_Driver and AMID_Disp must be specified for each mode. It is recommended that the entries for AMID_Dims, AMID_Disp, and optionally AMID_DriverData be specified when the Mode is first added.

DataBase Entries

For the most part, the DisplayInfo, DimensionInfo and NameInfo structures are unchanged. The MonitorInfo is slightly modified to reflect the graphics private RawMonitorInfo structure:

```
struct MonitorInfo
{
        struct  QueryHeader Header;
        struct  MonitorSpec  *Mspc;     /* pointer to monitor specification  */
        Point   ViewPosition;           /* editable via preferences          */
        Point   ViewResolution;         /* standard monitor ticks-per-pixel  */
        struct  Rectangle ViewPositionRange;  /* fixed, hardware dependent */
        UWORD   TotalRows;              /* display height in scanlines       */
        UWORD   TotalColorClocks;       /* scanline width in 280 ns units    */
        UWORD   MinRow;                 /* absolute minimum active scanline  */
        WORD    Compatibility;          /* how this coexists with others     */
        struct  Rect32  TxtRange;       /* \                                 */
        struct  Rect32  GfxRange;       /*  | Replaces the UBYTE pad[36]     */
        ULONG   pad;                    /* /                                 */
        Point   DefaultViewPosition;    /* original, never changes */
        ULONG   PreferredModeID;        /* for Preferences */
        ULONG   reserved[2];            /* terminator */
};
```

[All the current structures are terminated with a ULONG reserved[2], which was needed for the V37-V39 implementation of the database as a terminator. We may be able to dispose of these and save 8 bytes per structure per ModeID. Under V39, with all the monitors added, there are 264 ModeIDs. Not counting the NameInfos, that's 4 structures per entry, or 8448 bytes wasted as terminators! We will need to investigate the compatibility hit of s/w expecting so many bytes of data from a GetDisplayInfoData() call, but I don't see a real problem.]

There is, however, a small problem with the Txt/StdOScan values. These values are global to the whole Monitor, not just a specific ID. That means that the OScan values will need to be converted to values relevant for a specific ID. For example, on the standard PAL system, StdOScan may be 640x256 for hires, but 320x256 for Lores, and 1280x512 for SuperHires Laced. Therefore, the OScan values have to be stored in the greatest resolution possible, and scaled down.

The scaling factor comes from the DisplayInfo->Resolution value. GetDisplayInfoData() fills the Txt/StdOScan values of a DimensionInfo from the scaled Txt/StdOScan values in the MonitorInfo as:

```
dims->TxtOScan.MinX = (mntr->TxtOScan.MinX / disp->Resolution.x);
```

similarly for the other entries.

The struct VecInfo, which was introduced in V39 for graphics' private use should be replaced with a struct DriverInfo.

```
struct DriverInfo
{
        struct  QueryHeader   Header;
        APTR    DriverBase;             /* Points to the driver base for this
                                         * ModeID.
                                         */
        APTR    Data;                   /* Points to driver specific info */
        UWORD   pad[3];                 /* alignment - could be removed */
        ULONG   reserved[2];
};
```

We may also want to add another type, a struct SpriteInfo. The current information regarding sprites in the database is not too informative.

New display database information available:

Pixeltype ID - 4 character identifier for pixel-type. Identifies "weird" modes:

'HAM ' upper two bits of pixel index determine operation. Lower n-2 bits are RGB or index data.
'EHB ' High bit selects either rgb[index] or rgb[index]/2.
'TRUE' true-color. No palette.
'PLUT' pack lut mode.
'PRGB' packed rgb mode.

Whether or not a particular mode is "Displayable" at all. DIPF_ flag.

A "skeleton" bitmap pointer is also available from the database for each mode. This gives you something to pass as the friend to allocbitmap.

The DisplayInfo structure will be updated accordingly.

```
struct DisplayInfo
{
        struct  QueryHeader Header;
        UWORD   NotAvailable;   /* if NULL available, else see defines */
        ULONG   PropertyFlags;  /* Properties of this mode see defines */
        Point   Resolution;     /* ticks-per-pixel X/Y                 */
        UWORD   PixelSpeed;     /* approximation in nanoseconds        */
        UWORD   NumStdSprites;  /* number of standard amiga sprites    */
        UWORD   PaletteRange;   /* OBSOLETE - use Red/Green/Blue bits instead */
        Point   SpriteResolution; /* std sprite ticks-per-pixel X/Y    */
        UBYTE   pad[4];         /* used internally */
        UBYTE   RedBits;        /* number of Red bits this display supports (V39) */
        UBYTE   GreenBits;      /* number of Green bits this display supports (V39) */
        UBYTE   BlueBits;       /* number of Blue bits this display supports (V39) */
        TEXT    PixelType[4];   /* 4-character identifier for pixel type. */
        UBYTE   pad2;           /* find some use for this. */
        struct  NaturalBitMap *FriendBitMap;
                                /* Use as a Fiend BitMap to pass to AllocBitMap()
                                 * when opening the first ViewPort of this ModeID.
                                 */
};

#define DIPF_NOT_DISPLAYABLE 0x00400000
```

Driver entry points:

The following functions are all related to Views and ViewPorts, and need to be vectored by the driver. Most of these take the same parameters as their LVO counterparts. Differences are described.

/drv_ObtainDBufInfo(vp,dbi) - attach and initialize any driver specific information for double buffering.

/drv_ReleaseDbufInfo(dbi) - do any double buffering cleanup

/drv_ChangeVPBitMap()
/drv_FreeVPortCache() - this frees up any information cached by the driver at MakeVPort() and MrgCop() time. RTG equivalent of _LVOFreeVPortCopLists().

/drv_RefreshColors() - Updates all the colors for the passed ViewPort from the ColorMap. Should query colors using GetRGB32.

/drv_LoadView() - Load the view. The transition should be as clean as possible. drv_LoadView(NULL) should blank the screen.

/drv_UnloadView() - tells the driver that the it's View is no longer active, and can thus free no longer needed resources.

/drv_MakeVPort() - Builds up all the intermediate data it needs to display a single ViewPort.

/drv_MakeView() - Takes all the intermediate data of all the ViewPorts of the View, to build the final data it needs to show the whole View. RTG equivalent of _LVOMrgCop().

CopLists and CprLists allocated by MakeVPort and MrgCop will be tagged with respect to device, so that FreeCopList and FreeCprList can work.

/drv_FreeCopList(cop) - free cache information from MakeVPort.

/drv_FreeCprList(cpr) - free view info from MrgCop.

/drv_MoveSprite()
drv_ChangeSprite()
/drv_UpdateVPort() - should take account of changes to BitMap pointer, rxyoffset. RTG equivalent to _LVOScrollVPort().

/drv_WaitBeam() - used for WaitBeam and WaitBOVP.

/drv_AsyncWaitBeam() - async wait beam. Both WaitBeams return -1 if syncing was impossible.

/drv_CalcIVG()
/drv_WaitTOF() - new entry will need to take a View

?/drv_VBeamPos() - new entry will need to take a View?

/drv_GetSpriteInfoA() - returns the number of sprites that can be displayed on the passed ViewPort.

/drv_ValidVTAGSupported - checks whether a particular VideoControl tag is supported by the device.

/drv_VideoControl() - graphics first handles system videocontrol() functions (like ATTACH_CM_SET), and then passes the tag list onto drv_VideoControl. drv_VideoControl should invalidate the VTAG_IMMEDIATE variable.

Definitions of New Driver Functions

Driver entry points:

(/ indicates mandatory entries) all functions are passed a5=Struct GfxDeviceDriver * and a6=struct GfxBase *, unless otherwise noted. Functions must follow standard library function calling conventions unless otherwise noted.

```
void drv_RefreshColors(struct ViewPort *vp);
```

On the Amiga chips, this updates the color instructions in the intermediate and system copperlists with the contents of the ViewPort's ColorMap. On a 3rd party device, this will update the intermediate data and the hardware registers responsible for colors with the contents of the ViewPort's ColorMap.

```
void drv_KillView(struct MonitorSpec *mspc);
```

This tells the MonitorSpec that was associated with a View that it is no longer being displayed. This allows the driver to free any resources it may need only while the View is active. For example, the A2024 driver removes the vertical blank interrupt handler when it is no longer active with it's KillView() entry. This function should only be called by LoadView().

New Layers LVO

```
inout=FindLayerCR(struct RastPort *rp,LONG x, LONG y,
        struct Rect32 *cliprect, struct Point32 *offset,
        struct BitMap **bm)
```

Layer must be locked. Find a point in the layer. Set *cliprect to the bounds of the cliprect. Set *bm to point to the correct bitmap. If the point is outside of the cliprect list, set *cliprect to an exclusion rectangle it the pointer is non-null.

Layers will also have to grow support for the new rastport region stuff, both in FindLayerCR and DoHookClipRects.

Definitions of new LVOs

These new LVOs also have corresponding drv_ functions.

```
void NewWaitTOF(struct View *view, [struct Message *msg]);
void drv_WaitTOF(struct View *view, [struct Message *msg]);
```

This will wait until the beam has reached the top of the display. ?how do you find the view pointer for your intuition screen?

```
LONG NewVBeamPos(struct View *view);

LONG drv_VBeamPos(struct View *view);
```

This one is still only of use when the caller knows it won't be preempted. Also, the current _LVOVBeamPos() in graphics.library doesn't work properly anyway. This new one will! Returns -1 if board is incapable of doing it.

```
BOOL /drv_WaitBeam(struct View *v, struct ViewPort *vp, ULONG beamposy);
```

Return when the beam is at the given position (approximately). May break a forbid. May use polling. If the device can't beam synchronize, return false. Can automagically adapt to CPU speed?

```
BOOL /drv_ASyncWaitBeam(struct View *v, struct ViewPort *vp, ULONG beamposy, struct
Message *msg);
```

Reply the message when the beam reaches a certain position. May fail.

```
void GetSpriteInfoA(struct ViewPort *vp, struct TagList *tags);

void drv_GetSpriteInfoA(struct ViewPort *vp, struct TagList *tags);
```

Many applications need to know how many sprites they can use in a given ViewPort, but the database only tells them how many sprites the hardware can support at most. This new call should return more useful information.

TAGS:

MS_Origin - How many sprites can be used when this ViewPort is at it's origin position. ti_Data points to a ULONG to store the result (yes, *real* tags in graphics).

MS_WorstCase - The worstcase number of sprites that would be available (eg on the Amiga, worst case would be while scrolling the ViewPort). ti_Data points to a ULONG to store the result

MS_WithTricks - The worstcase number of sprites that would be available if tricks were pulled. eg, on the Amiga, we could drop the bandwidth to increase the number of sprites. ti_Data points to a ULONG to store the result

MS_ReUse - If the device supports reusing of sprites, then the result is the number of lines needed between each sprite reuse. If sprites cannot be reused, the result is -1. eg, the Amiga would return 1. ti_Data points to a ULONG to store the result

Getting Started

Call GetMonitorNumber() with the Monitor number allocated by C= for your card. Use the resultant monitor number in all the ModeIDs for the monitor.

Create the Driver.
Initialize the MonitorSpec.
Call CreateMonitor().
Initialize and call AddModeID() for each mode in the monitor.

Easy!

Rough Example

```
#define MONITOR_NUM 0x40010000
#define MONITOR_NAME "MyMonitor"
#define MONITOR_PREFERREDID ???
#define MONITOR_FLAGS 0
#define MONITOR_TOTROWS ???
#define MONITOR_TOTCC   ???
#define MONITOR_COMPAT MCOMPAT_NOBODY
#define MONITOR_MINROW  ???
#define MONITOR_DATASIZE ???

struct MonitorSpec *mspc;
APTR DriverBase;
ULONG MonitorNum = MONITOR_NUM;
UBYTE MonitorData[MONITOR_DATASIZE];

struct TagItem DriverTags[] =
{
        {DRV_InitVectorTable, VectorTable},
        {TAG_END},
};

struct TagItem MonitorTags[] =
{
        {AM_MS_Flags, MONITOR_FLAGS},
        {AM_Compat, MONITOR_COMPAT},
        {AM_MinRow, MONITOR_MINROW},
        {AM_DataSize, MONITOR_DATASIZE},
        {AM_DriverData, MonitorData},
        {TAG_END},
};

struct NewMonitor nm =
{
        &MonitorNum,
        MONITOR_NAME,
        NULL,
        NULL,
        MONITOR_PREFERREDID,
        MONITOR_TOTROWS,
        MONITOR_TOTCC,
        NULL,
        /* LegalView Here. */,
        /* View Resolution here */,
};

/* Create the Driver */
if (DriverBase = CreateDriverA(DriverTags))
{
        /* initialize the NewMonitor */
        nm.Driverbase = DriverBase;
        nm.DriverNumber = GetUniqueID();

        /* Add the Monitor */
        if (mspc = CreateMonitorA(&nm, MonitorTags))
        {
                /* CreateMonitorA() may have assigned a different monitor number
```

```
            * and name.
            */
           printf("Monitor Number assigned is 0x%lx\n", MonitorNum);
           printf("Monitor name is %s\n", mspc->ms_Node.xln_Name);

           /* for each ModeID in the monitor,
            * create the data, and call AddModeID()
            */
       }
       else
       {
           /* Whoops. Clean up time. */
           CloseDriver(DriverBase);
       }
}
```

DriverBase

The DriverBase works in much the same way as a LibraryBase. The positive offsets from the base are for driver data, and the negative offsets are function pointers.

ViewPortExtra

The ViewPortExtra structure was extended for V39.

```
struct ViewPortExtra
{
        struct ExtendedNode n;
        struct ViewPort *ViewPort;      /* backwards link */
        struct Rectangle DisplayClip;   /* makevp display clipping information
        /* These are added for V39 */
        APTR   VecTable;                /* Private */
        APTR   DriverData[2];
        UWORD  Flags;
        Point  Origin[2];               /* First visible point relative to the
                                           DClip.
                                            One for each possible playfield.
                                         */
};
```

VecTable points to the same VecTable that the DriverInfo for the ViewPort's mode points to. DriverData[] are pointers to driver specific data, and are free for the driver to use. Flags is for the system's.

We no longer need the Origin[]. We could remove these, and replace them with .

```
APTR DriverBase;
ULONG DriverFlags;
```

How to get at the driver vector table from a given structure

RastPort	through the BitMap
BitMap	if bm_BytesPerRow odd, than has DriverPtr, else Default.
View	Either the topmost viewport, or the ViewExtra
ViewPort	through the ColorMap or RasInfo->BitMap
ColorMap	through the ViewPortExtra
ViewPortExtra	contains a pointer to it.

```
struct StandardBitMap
{
    UWORD   BytesPerRow;        /* odd means a NaturalBitMap */
    UWORD   Rows;               /* height */
    UBYTE   Flags;              /* reserved for graphics */
    UBYTE   Depth;              /* bits per pixel up to 255 */
    UWORD   pad;                /* reserved for graphics */
    PLANEPTR Planes[];          /* plane pointers */
};
```

If the bm_pad is equal to a magic value, than the flags field is valid. Flags include interleaved, true-color, fastmem, etc. A colormap can be associated with a StandardBitMap (stored in Planes[Depth]) via SetBitMapAttrs. This is so that standard indexed bitmaps can be blitted to true-color screens. When a bitmap is passed as a friend to AllocBitMap, its ColorMap is installed into the new bitmap.

An embedded StandardBitMap is created by calling AllocBitMap with the appropriate flags and then calling SetBitMapAttrs to attach the planes.

A StandardBitMap is distinguished from a NaturalBitMap by bit 0 of BytesPerRow. Bit 0 is set for a NaturalBitMap.

A true-color bitmap has depth/3 planes for each gun. (or maybe depth?)

```
struct NaturalBitMap    // a bitmap for a foreign device driver
{
        UWORD nb_GfxFlags;      /* graphics flag. Bit 1 always set */
        UWORD nb_Rows;          /* height */
        UBYTE nb_Flags;         /* true-color, etc. */
        UBYTE nb_Depth;         /* bits per pixel, up to 255 */
        UWORD nb_pad;           /* set to magic number */
/* fields after this do not match a normal bitmap structure */
        ULONG nb_NewFlags;      /* flags for gfx use. */
        struct GfxDeviceDriver *nb_Driver;
                                /* pointer to device driver */
        struct ColorMap *nb_AssociatedColorMap;
                                /* colormap associated with
                                   this display bitmap */
        ULONG   nb_FutureExpansion[2];
                                /* future for graphics */
        driverdata[];           /* variable length device information */
};


struct RastPort
{
    struct  Layer *Layer;
    struct  BitMap   *BitMap;   /* could now be a NaturalBitMap */
    UWORD  *AreaPtrn;           /* ptr to areafill pattern */
    struct  TmpRas *TmpRas;     /* obsolete for non-amiga drivers, but should
                                   definitely not be re-used */
    struct  AreaInfo *AreaInfo; /* AreaInfo ptr for all devices */
    struct  GelsInfo *GelsInfo; /* yuck, gross. why? */
    UBYTE   Mask;               /* driver private for >8 bit drivers */
    BYTE    FgPen;              /* driver private for >8 bit drivers */
    BYTE    BgPen;              /* driver private for >8 bit drivers */
    BYTE    AOlPen;             /* driver private for >8 bit drivers */
    BYTE    DrawMode;           /* standard drawmode */
    BYTE    AreaPtSz;           /* 2^n words for areafill pattern. */
    BYTE    linpatcnt;          /* current line drawing pattern preshift */
    BYTE    dummy;              /* graphics private. use to store last charfor kanji */
    UWORD  Flags;               /* miscellaneous control bits. Graphics private */
    UWORD  LinePtrn;            /* 16 bits for textured lines */
    WORD   cp_x, cp_y;          /* driver private. Drivers supporting 32 bit coordinates
                                   will want to store these elsewhere */
    UBYTE   minterms[8];        /* driver private!! */
    WORD   PenWidth;            /* reserved for graphics.library */
    WORD   PenHeight;           /* reserved for graphics.library */
```

```
struct  TextFont *Font;     /* current font address */
UBYTE   AlgoStyle;          /* the algorithmically generated style */
UBYTE   TxFlags;            /* text specific flags */
UWORD   TxHeight;           /* text height */
UWORD   TxWidth;            /* text nominal width */
UWORD   TxBaseline;         /* text baseline */
WORD    TxSpacing;          /* text spacing (per character) */
APTR    *RP_User;           /* user data */
ULONG   longreserved[2];    /* reserved for graphics */
UWORD   wordreserved[7];    /* reserved for driver */
UBYTE   reserved[8];        /* reserved for graphics */
    /*
            gfx expansion bytes: 24
            extra bytes available to 8 bit driver: 23
            extra bytes for 24 bit driver: 17
            extra bytes for 24 bit driver with 32 bit coordinates: 13
    */
```

Low-Level Rendering Operations

```
BOOL /drv_GetBitMapAttr(bm,attr,&data)
```

return the given attribute for a bitmap. returns whether or not the driver knows the given attribute.

```
BOOL /drv_SetBitMapAttr(bm,attr,data)
```

set the given attribute for a bitmap. returns whether or not the driver knows the given attribute.

```
struct BitMap * /drv_AllocBitMap(sizex,sizey,depth,flags,friend)
```

Allocates a bitmap.

```
/drv_ReadPixData(bm,x,y,w,h,&buffer)
```

Read pixels from a rectangle in a bitmap. The storage size will be (bm->depth+7)/8 bytes per pixel. The driver may store the data in whatever manner it chooses, as graphics.library will always treat it as whole pixels.

```
/drv_WritePixData(bm,x,y,w,h,,&buffer)
```

Write pixels to a rectangle in a bitmap. The storage size will be the same as ReadPixData. ReadPixData and WritePixData can be used to implement all blits and scaling blits.

```
/void drv_WaitBlit(void)
```

Wait for any previously queued graphics operations to complete. May NOT break a FORBID!!. WaitBlit on the standard HW.

```
/void drv_Flush(void)
```

Wait for any previously queued graphics operations with this driver to complete. Must be called before FreeBitMap. MAY break a FORBID!!. WaitBlit on the standard HW.

```
/drv_SetRastBM(struct NaturalBitMap *bm,struct RastPort *rp,minx,maxx,miny,maxy,color)
```

set the given rectangle in the bitmap to the given indexed color. When blitting to truecolor, will do a color lookup.

```
/drv_BltTemplateBM(UWORD *src,srcx,srcmod,DestRPort,destx,dsty,sizex,sizey,destbm)
```

Blit template to a bitmap. Should obey RastPort parameters.

```
/drv_MatchTemplateBM(rp,struct BitMap *bm,srcx,srcy,dstx,dsty,width,height)
```

Fill plane 0 of the specified rectangle of bm with 1's wherever the pixels match the APen setting of the rastport. Used by Flood and highlight mode.

```
struct DDA {
        FUNCPTR AdvanceDir[8]    ; ptrs to functions for advancing
                                 ; U UR R DR D DL L UL
        FUNCPTR DrawAdvanceDir[8]
                        ; ptrs to functions for drawing and advancing
                                 ; U UR R DR D DL L UL
        FUNCPTR Move             ; move with no draw. Resets based on BitMap (a1).
        ULONG driverdata[?];     ; DDA context. The DDA context also consists of
                                 ; a0 and d0.
}

/err=drv_InitDDA(struct DDA *dda,struct RastPort *rp,struct BitMap *bm,x,y)
```

Initialize for pixel drawing, based upon rastport settings. Must fill in vector pointers. Can use An and Dn for temps across the lifetime of a DDA. Multiple DDA's must not interfere with each other (meaning that typically they will have locking equivalent to OwnBlitter)

```
/drv_FinishDDA(struct DDA *)
```

free up any resources associated with a DDA.

```
/drv_WritePixelBM(struct BitMap *bm, struct RastPort *rp,x,y)
```

Write a pixel into a bitmap, using current context settings in rastport.

```
ULONG /drv_ReadPixelBM(struct BitMap *bm, struct RastPort *rp,x,y)
```

Read a pixel from a bitmap. Straight ReadPixel is not too useful on a true-color device. Most Set/Get rastport calls map to direct drv_xxx calls, and are mandatory.

```
/drv_ReadPixelRGB8ArrayBM(bm,xl,yl,w,h,&rgb32)
```

Get 8-bit rgb values for a pixel array. Even works in HAM mode.

```
/drv_WritePixelRGB8ArrayBM(bm,xl,yl,w,h,&rgb32)
```

Write 8-bit rgb values for a pixel array. Only works for true-color modes.

```
/drv_WritePixelArrayBM(bm,xl,yl,w,h,&rgb32)
```

Write (depth+7)/8-bit index values from a pixel array.

```
/drv_ReadPixelArrayBM(bm,xl,yl,w,h,&rgb32)
```

Read (depth+7)/8-bit index values from a pixel array.

```
/drv_Move(rp,x,y)
```

Move the cursor position.

Medium Level Primitives

```
drv_DrawLine(rp,bm,x1,y1,x2,y2)
```

Draw a line, unclipped

```
drv_DrawClippedLine(rp,bm,x1,y1,x2,y2,&rect32)
```

Draw a line, clipped to the given rectangle. Must use perfect raster clipping, and handle the pattern shift properly.

```
drv_Bltxxx(srcbm,srcx,srcy,destx,desty,width,height,msk,op)
```

BitBltMap family of functions. The default BltBitMap demultiplexes the various combinations and calls one of the following:

NaturalToNatural
StandardIndexedtoNatural
StandardTrueColorToNatural
NaturalToStandardIndexed
NaturalToStandardTrueColor

```
drv_BltMaskBitMapBM(srcbm,srcx,srcy,destbm,destX,destY,sizeX,sizeY,op,bltmask)
```

Perform masked blit. Default entry will do multiple BltBitMaps.

```
drv_FilledEllipse
```

draw a filled ellipse, using areafill rules. This will be called by AreaEnd when there is only on ellipse in the area buffer

```
drv_FilledPoly
```

draw a filled (potentially concave) polygon.

```
/drv_BltPatternBM(rp,mask,xmin,ymin,xmaxm,ymax,maskbpr,&bm)
```

Do a pattern blit, using rastport settings.

```
/drv_FillYLRBM(rp,xoffset,yoffset,&ylrarray,&bm,&cliprect,patxoffset,patyoffset)
```

Fill a series of horizontal lines, using pattern fill rules. &cliprect may be nill (meaning no clipping). the ylrarray is an array of ULONG triplates, terminated by -1. Many of the higher level calls which call this will generate the array in top to bottom order, so it is worth optimizing for this case (though you can't depend on it).

High Level Primitives

```
drv_Flood(rp,mode,x,y)
drv_BltBitMap()
drv_DrawEllipse(rp,xcenter,ycenter,a,b)
drv_ReadPixelRGB32(rp,x,y,&rgb32)
```

Read pixel data out. handles indexed color or true-color. handles HAM! Rp Entry is needed for HAM support through layers.

WritePixel example:

```
#define BMTODRIVER(bm) \
        (bm->BytesPerRow & 1)?
         ((struct NaturalBitMap *)bm)->bm_GfxDriver
         :GBASE->DefaultDriver

CALLOFFBITMAP is impossible to #define in C, but is doesn't matter for this example.

WritePixel(rp,x,y)      // _LVOWritePixel points here.
{
        if (rp->Layer)
        {
                struct BitMap *found_bitmap;
                COORD32 xyoffsets;
                LOCKLAYER(rp->Layer)
                if (FindLayerCR(rp,x,y,NULL,&xyoffsets,&found_bitmap))
                {
                    CALLOFFBITMAP(found_bitmap,OFS_WRITEPIXELBM)(rp->BitMap,rp,
                        x+xyoffsets.x,y+xyoffsets.y,BMTODRIVER(found_bitmap));
                }
                UNLOCKLAYER(rp->Layer);
        }
        else
        {
                CALLOFFBITMAP(found_bitmap,OFS_WRITEPIXELBM)(rp->BitMap,rp,x,y,
                        BMTODRIVER(found_bitmap));
        }
}


void drv_WritePixelBM(register __a0 struct myBitMap *mybm,      // NaturalBitMap
ptr
                        register __a1 struct RastPort *myrp,
                        register __d0 LONG x, register __d1 LONG y,
                        register __a5 struct MyLibBase *MyLibBase)
{
        ObtainSemaphore(MyLibBase->BlitSemaphore);
                        /* not really needed for atomic operation */
        UBYTE *pixeladr;
        UBYTE pixdata;
        UBYTE msk=rp->Mask;

        pixdata=(rp->DrawMode & INVERSEVID)?rp->FgPen:rp->BgPen;
        pixdata &= msk;                         /* for easy ORing in */
        pixeladr=mybm->BytesPerRow*y+x;
        switch((rp->DrawMode) & ~INVERSEVID)    /* simple chunky device.
                                                   Uses normal DrawMode
                                                   in rastport for SetDrMd
                                                   storage */
        {
                case JAM1:
                case JAM2:
                        *pixeladr= *(pixeladr & ~msk) | pixdata;
                        break;
                case COMPLEMENT:
                        *pixeladr ^= msk;
        }
}
```

True color blit rules:

Bitmaps allocated as friends of other bitmaps inherit their colortables.

indexed->indexed: no translation

indexed -> true color

```
Src bitmap      dest bitmap
==========      ===========
no table        yes table       uses dest table
yes table       yes table       assumes src table
yes table       no table        uses src table
no table        no table        uses default table for source
```

true-color-> true color

no colortable needed

General Graphics Enhancements & Misc RTG Issues

New graphics videocontrol tag: VTAG_INTERMED_UPDATE : When this is set for a viewport, graphics calls which change the copper lists need not update the information cached by MakeVPort (the intermediate copper lists). Only the information (hardware copper lists) generated by MrgCop need be updated. This applies to LoadRGB32, ScrollVPort, ChangeUCL, VideoControl with VTAG_IMMEDIATE, and ChangeVPBitMap. Basically you are telling graphics that you will not be MrgCop'ing again without remaking the viewport. Should help games/multimedia a lot.

Requires intuition support to protect screens from RethinkDisplay.

VideoControl will either stop barfing on bad tags, or will be replaced.

VideoControl will grow a new tag for turning "ScreenFlash" on and off. This will be used for ScreenBeep().

New alloc/free structures will be handled via GfxNewTags and GfxFree to stop LVO overpopulation.

```
GfxNewTagList(gfxnodetype,tags)
```

aliased to GfxNew.

Point transformations:

```
struct Transformation *GfxNewTags(GFXTYPE_TRANSFORM,tags)
```

allows transformation from real-world units to screen units, rotation, scaling, translation, and shearing. Transformations can be concatenated by passing an old transformation.

```
TransformPoints(struct Transformation *tf,&srcpts,&dstpts,n)
```

src and dest can be same.

```
InverseTransformPoints(struct Transformation *tf,&srcpts,&dstpts,n)
```

src and dest can be same.

graphics clipping will be made robust out to 16 bits (for the motherboard driver). The actual primitive implementation will be robust to 32 bits for other drivers.

?What to Do About HAM10?

GetRGB32 will return the system default colors is the passed ColorMap is NULL. This is for support for true-color bitmaps when there is no colormap attached.

PenWidth and PenHeight will be supported for Draw, DrawEllipse, and DrawBezier.

SetRPAttrs can be used to set the true-color to render in on true-color devices.

```
FillYLR(rp,xoffset,yoffset,&ylrarray)
```

Fill a series of horizontal lines, using BltPattern rules. Calls drv_FillYLRBM(rp,xoffset,yoffset,&ylrarray,&bm,&cliprect).

```
SetBitMapAttrsA(bm,attrs)
```

Change bitmap attributes. Will be used to install a colormap into a bitmap.

```
DrawBezier(rp,&points)
```

Draw a bezier curve (or a series of them?)

```
ElipticalArc(rp,points)
```

Draw an elliptical from 3 points.

```
?RemapBitMap(&srcbm,&destbm,x,y,w,h,&colorpens,...)?
RemapBitMapRastPort(&srcbm,&destrp,x,y,w,h,&colorpens)
```

Remap a bitmap. Can convert true-color/ham to indexed. Can convert indexed to true-color, etc. ColorPens is a handle to a penarray to free via GfxFree.

?AreaBezier?

ObtainBestpen will allow specification of a color to avoid. This is to avoid the catastrophe of black-on-black text. It will also gain the ability to give back "n" colors at once (each with an associated weight), which will improve remap quality.

Bitmaps will work in virtual memory / fast ram

SetRPAttrs can be used to set a clipping region and (maybe) origin for a RastPort. This is similar to InstallClipRegion, but does not modify the layer's clip rectangles, and works even if they change out from under it. Installing the region will be inexpensive, but drawing through it will be slower than the InstallClipRegion method. The other advantage of the rastPort clipping region is that it works on non-layered rastports, and that there can be multiple independent ones per layer. I would like to re-

implement regions for this release, in order to:

(a) Use exec memory pools
(b) keep them y/x sorted
(c) Add the (future or present) capability for bitmapped or other types of regions.
(d) make it possible in the future to generate them by calling graphics primitives (like on the Mac).
(e) allow sharing of spans and rectangles between multiple regions via reference counting.

SetRPAttrs will also be able to set the pen width and height and other style tags.

Smart fonts:

A Smart font knows how to render itself. Hanging off of the font extension is a pointer to a vector table. The following vectors are present:

```
OpenFont()
CloseFont()
DrawText(textparams)
        inittext(params)
        RenderTemplate(temp,rp,string,bounds)
        donetext(params)

TextLength()
TextFit()
TextExtent()
```

and also vectors for diskfont.library:

```
MakeScaledFont(params)

SmoothPoly(rp,&points,&colors,npoints,[lookuptable])
```

Gouraud shade a convex polygon. lookuptable is for indexed-color systems, and is optional. If no lookuptable is specified on an indexed color system, actual color indices will be interpolated. A simple ordered dither is supported.

```
BOOL RectInRegion(struct Region *,struct Rectangle *)
```

Returns true if there is any possible intersection between the rectangle and region.

```
BltBitMapTags(srcbm,x,y,destbm,x,y,width,height,tags....)
BltBitMapRastPortTags(srcbm,x,y,destrp,x,y,width,height,tags....)
```

New style blit. Supports AAA and true-color arithmetic modes: defaults to copy.

BBMTAG_Mode:

```
BBM_ADD         constants can be added via subtraction.
BBM_ADDS        add with saturate
BBM_SUB(S)
BBM_BLEND       blending %
BBM_OR
BBM_AND
BBM_COPY
.etc.
```

BBMTAG_MaskBitMap, &standardbitmap. Arbitrary depths are supported for blending.

A device driver can be written which would act to store displaylists, which would create the ability for simple printing, resolution independence, etc.

```
DeviceSpecificFunction(funcnum,tags)
```

fucnums 0-7fffffff reserved for standardization.

```
IsSupported(funcnum,bitmap)
```

ChangeUCL:

```
******* ChangeUCLA ************************************************************
*
*   NAME
*       ChangeUCLA -- Modify a viewport's user-copperlist
*       ChangeUCL  -- varargs stub fro ChangeUCLA
*
*   SYNOPSIS
*       status= ChangeUCLA(vp, regnum, regmask, tags)
*                          a0     d0      d1       a1
*
*       LONG ChangeUCLA(struct ViewPort *vp, ULONG regnum,
*                       ULONG regmask, struct TagItem *);
*       LONG ChangeUCL(struct ViewPort *vp,ULONG regnum,
*                       ULONG regmask, tag1,...);
*
*   FUNCTION
*       To Modify (or optionally read) register values from a viewport's
*       copper list.
*
*   INPUTS
*       vp      - pointer to a viewport.
*       regnum  - address of custom chip register to modify MOVE for in
*                 the copper list
*       regmask - mask of bits to be changed.
*       tags    - standard tag list. The following tags are currently supported:
*
*               UCL_ReadValue: Reads the current value from the copper list into
*                       the long variable pointed to by ti_Data.
*               UCL_WhichWait: Specifies how many CWAITs in your user copper list
*                       to skip over before looking for the register. Defaults
*                       to 0.
*               UCL_WhichReg: Specifies how occurrences of the given register
*                       to skip over (after UCL_WhichWait is satisfied). Defaults
*                       to 0. It will stop when it finds the corrent occurrence
*                       of the register, or at the next wait.
*               UCL_NWaits: specifies how many CWAIT/CMOVE groups to apply the
*                       given changes to.
*               UCL_NRegs: specifies how many occurrences of the appropriate register
*                       to change in each CWAIT/CMOVE group.
*               UCL_CachePtr : Allows The system to cache the locations of
*                       user-copper list instructions. You should pass a pointer to
*                       a void * pointer in the ti_Data. When your program is done, it
*                       should FreeVec the cache information pointed to by the
*                       pointer.
*   RESULT
*       0 = function not implemented (value returned by V39)
*       <0 = an error occurred
*       >0 = success.
*
*   NOTE:
*       This function is hardware dependent, and only works on standard Amiga-
*       compatible chips.
*******************************************************************************
```

◆

DSP On The Amiga

Using The AT&T DSP3210™ and VCOS™ On The Amiga

Document Revision 0.6

1993 DevCon Release

by
Eric Lavitsky
December, 1992

Copyright © 1992 Lavitsky Computer Laboratories, Inc.
All rights reserved worldwide.

This document contains preliminary information and is subject to change without notice.

Disclaimer: While the author(s) of this document have made every attempy to verify its accuracy, the information provided herein is provided "as is" without warranty of any kind, either express or implied. The author(s) and Lavitsky Computer Laboratories, Inc. assume no liability for any damages, direct or indirect, resulting from any defect in this information.

This document assumes an understanding of the Amiga architecture and programming practices.

Amiga is a registered trademark of Commodore-Amiga, Inc. VCOS and DSP3210 are trademarks of AT&T. MS-DOS is a registered trademark of Microsoft.

This document was created entirely on an Amiga 3000 using PageStream 2.2 from Soft-Logik.

I. Introduction

As the demand for high quality digital data in multimedia applications increases, so does the demand to rapidly process the accompanying complex data streams. Existing personal computer CISC and RISC architectures lack the signal processing throughput required to process these data streams efficiently and in real time. Digital Signal Processors (DSPs) are designed to process real-world signals (e.g., audio and video signals) in real time. Recently, the cost of DSPs has dropped to the point that they can be integrated into personal computer systems very effectively. Even in more advanced RISC and workstation systems, DSPs can be used to perform signal processing tasks without degrading the performance of the host CPU.

The Amiga has always been a strong multimedia computer, in fact, many would consider it the first viable multimedia personal computer. Its hardware and software architecture is well suited to processing multiple simultaneous data streams. Amiga designers and system engineers have always had the foresight to know that the existing system would need to grow in order to meet the future needs of its users. This foresight has made it relatively easy to integrate a DSP into the Amiga system.

Many system manufacturers have attempted to integrate DSPs into their platforms in the past, but have met with only limited success. The key factors contributing to this have been the limitations of the DSPs they have used and the lack of flexible and advanced applications that take advantage of them. The AT&T DSP3210 is a very flexible and low-cost DSP capable of computing up to 33 million floating point operations per second (MFLOPS). It can be designed into a host computer with relatively little overhead. AT&T has solved the lack of applications by providing a multitasking operating system called VCOS, as well as a full software library implementing a wide variety of signal processing and multimedia algorithms for the DSP3210. Using the AT&T DSP3210 and VCOS, it is now possible to provide applications such as speech recognition, CD-quality audio (compressed), high-speed telecommunications, high-quality music synthesis and many others with little or no impact on system performance. Even a powerful CPU like the Motorola 68040 could not hope to keep up with the processing demands of even a single one of these applications, let alone several of them simultaneously.

II. Hardware

II.I Overview

The DSP3210 is a full 32-bit floating point DSP implemented in .9 micron CMOS. It provides many advantages over fixed point DSPs such as the Motorola 56000. Some of the main features of the DSP3210 include:

- 32-bit floating point arithmetic.
- 32-bit addressing.
- Large (8K) on-chip, zero wait-state memory.
- Single cycle instructions (for up to 33Mflops).
- Share bus with Motorola or Intel style CPU.
- Serial I/O with DMA transfer counters for up to 25Mbits/second transfer:
  Serial data transfers occur without processor intervention.
  Cycles are stolen when necessary.
  DMA control for serial in and serial out.
- Barrel Shifter for bit manipulation in graphics or data encryption.
- Both mu-law and A-law encoding.
- Bit I/O general purpose 8-bit I/O port for control of external hardware

Copyright â 1992, Lavitsky Computer Laboratories, Inc.

- Programmable 32-bit timer for interval timing, rate generation, event counting or waveform generation.
- Fully vectored interrupt structure with hardware context save:
  Allows very fast interrupt processing, up to 2 million/second.
- Low power CMOS design.

No special programming is required on the DSP3210 to implement floating point algorithms, or to process signals with a much larger dynamic range (in excess of 1500db as opposed to < 300db for fixed point). The DSP3210 is also designed to share a host memory bus with either a Motorola or Intel style CPU. This greatly reduces system cost by removing the requirement for expensive fast local memory for the DSP. This also removes any practical restrictions on program or data size. A large on-chip cache (8K) combined with software that intelligently utilizes the cache allows the DSP3210 to execute complex signal processing algorithms without expensive local memory. All instructions execute in a single cycle (four clock periods: 80ns for a 50Mhz part or 60ns for a 66Mhz part) and includes all floating point post normalization (which is performed automatically). A single instruction may have two floating point operations: a floating point multiplication and a floating point addition. The DSP3210 also supports up to four memory accesses in a single instruction cycle (quad-word transfer). The DSP3210 architecture features seven functional units:

- Control Arithmetic Unit (CAU)
- Data Arithmetic Unit (DAU)
- On-chip memory (RAM0, RAM1, Boot ROM)
- Bus Interface
- Serial I/O (SIO)
- DMA Controller (DMAC)
- Timer/Status Control (TSC)

II.II The Control Arithmetic Unit

The CAU is responsible for address calculation, branching control and all 16 and 32-bit integer logic and arithmetic operations. It is a RISC core consisting of a 32-bit Arithmetic Logic Unit (ALU), a 32-bit Program Counter (PC), 22 32-bit general purpose registers (r0-r22) and a 32-bit barrel shifter. This core executes instructions at up to 16.7 million instructions per second. There are special register considerations in the CAU:

r0	hardwired to 0 (always)
r1-r14	DA instruction memory reference (X,Y,Z) pointer registers
r15-r19	DA instruction memory reference (X,Y,Z) increment registers
r20	used by error exception facility to store old pc
r21	stack pointer (sp)
r22	pointer to the exception vector table (evtp)

The CAU provides the following branching and control instructions:

if (COND) goto {N, rB, rB+N}	Conditional branch based on flags
if (rM-->=0) goto {N, rB, rB+N}	Conditional branch using loop counter
goto {N, rB, rB+N, M, rB+M}	Unconditional branch
nop	No operation
call {N, rB, rB+N, M}(rM)	Call subroutine
return {rM}	Return from subroutine
do K,{L, rM}	Do next K+1 instruction(s) L+1 or (rM+1)

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

	times. K=0,1,2...127; L=rM=0,1,2...2047
dolock K,{L, rM	Signals interlocked bus access
doblock {L, rM}	Signals quad-word transfers
ireturn	Return from interrupt
sftrst	Soft-reset; changes error level to base level; encoded as spc=(byte)r0
waiti	Wait for interrupt; encoded as spc=(long)r0

where: rB = pc, r0-r22
rM = r1-r22
N = 16-bit unsigned integer
M = 24-bit unsigned integer
COND = one of the DSP3210 condition codes (refer to DSP3210 manual)

II.III The Data Arithmetic Unit

The DAU consists of a 32-bit floating point multiplier, a 40-bit floating point adder, four 40-bit floating point accumulators (a0-a3), a clip test register (ctr), and a control register (dauc). The multiplier and adder operate in parallel to perform up to 16.7 million computations per second (12.5 million for a 50Mhz part) of the form (a=b+c*d), also known as a multiply-accumulate. The DAU contains a four stage pipeline which is visible to the application programmer. The DAU supports the following floating point formats:

Single precision (32-bit) in both DSP32 and IEEE format
Extended single precision (40-bit) (uses 8 mantissa guard bits)

Single instruction data type conversions are done in the DAU hardware:

DSP32 and IEEE 32-bit floating point
16/32-bit integer
8-bit unsigned
mu-law and A-law

The DAU has a number of special instructions to greatly simplify data type conversions and other common operations:

[Z=] aN = ic(Y)	Input conversion mu-law, A-law, 8-bit linear to float.
[Z=] aN = oc(Y)	Output conversion float to mu-law, A-law, 8-bit linear.
[Z=] aN = float16(Y)	16-bit integer to float.
[Z=] aN = float32(Y)	32-bit integer to float.
[Z=] aN = int16(Y)	Float to 16-bit integer (round or truncate, dauc[4]).
[Z=] aN = int32(Y)	Float to 32-bit integer (round or truncate, dauc[4]).
[Z=] aN = round(Y)	Round to nearest, float(40) to float(32).
[Z=] aN = ifalt(Y)	Conditional assignment/memory write.
[Z=] aN = ifaeq(Y)	Conditional assignment/memory write.
[Z=] aN = ifagt(Y)	Conditional assignment/memory write.
[Z=] aN = dsp(Y)	IEEE to DSP format conversion.
[Z=] aN = ieee(Y)	DSP to IEEE format conversion.
[Z=] aN = seed(Y)	32-bit to 32-bit reciprocal seed.

Where [Z=] indicates that condition codes may be set. Note that Y may not be a0-a3 for the *dsp()* special function.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

II.IV Addressing Modes

DSP3210 assembly language exhibits a syntax very similiar to 'C'. The notation conventions are as follows: a0-a3 are the accumulators (DAU), and r0-r22 are the CAU registers. Instructions take the following appearance:

```
r2 = (long)r1           ; CAU register direct: store the contents of r1 in r1
r1 = (long)*r1          ; store value pointed to by r1 in r1
r1 = (long)r1 + 1       ; increment r1 by 1
*r2++ = (long)r1        ; postmodify increment r2 after storing r1 there (in *r2)
r3 = (long)r1 + r2      ; add two numbers in r1, r2: store the result in r3
r3 = (long)*r1++r2      ; post modify increment r1 by r2: store the result in r3
a2 = a2 + *r2 * a3      ; use that pipeline!
```

The following table lists the various addressing modes supported by the DSP3210:

Addressing Mode	Instruction Type			
	CA Data Move Group (CAU Reg)	CA Data Move Group (I/O Reg)	CA Arithmetic/ Logic Group	DA M/A & Special Func
Short Immediate	Yes			
24-Bit Immediate	Yes			
Memory Indirect	Yes			
CAU Register Direct	Yes	Yes	Yes	
IO Register Direct	Yes			
DAU Register Direct				Yes
Register Indirect	Yes	Yes		Yes
Register Indirect with Postmodification	Yes	Yes		Yes

II.V Latency Issues

The most difficult aspect of programming the DSP3210 is being aware of latency in the instruction pipeline. There are four cases in the DAU when pipeline affects latency. The cases are:

1. *DA Memory Writes*. When a DA instruction specifies a write to memory, the value written is not available to be read from that location until four instructions later (a three instruction latency). For example:

```
*r3 = a0 = a0           ; Instruction 1
*r3 = a3 = a3           ; Instruction 2
.                       ; Instruction 3
.                       ; Instruction 4
a1 = *r3                ; Instruction 5
```


Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

The value read in instruction 5 is the value written in instruction 1, not instruction 2. Instructions 3 and 4 are latent instructions for instruction 1 and instructions 3, 4, and 5 are latent for instruction 2.

2. *Accumulator as Multiplier Input.* When an accumulator is used as an input to the multiplier, its value is established no sooner than three instructions prior to the multiply instruction (a two instruction latency). Note that this also applies to an accumulator using the X field of an instruction of the form:

[Z=] aN = [-]Y {+,-}X

For example:

```
a0 = a0 + *r1**r2       ; Instruction 1
a0 = a0 + a1            ; Instruction 2
.                       ; Instruction 3
a2 = a0 * a0            ; Instruction 4
a1 = a2 + a0            ; Instruction 5
```

The value of a0 used in instruction 4 is calulated in instruction 1. The value of a0 used in instruction 5 is calculated in instruction 4 since there is no latency effect on accumulators used as inputs to the adder.

3. *Branching.* When a CA Control Group instruction of the form *if()goto, call, return, goto* is executed, the instruction immediately following is also executed before the branch occurs. This is commonly referred to as a delayed branch. The *ireturn* instruction is different, and execution of the base-level program resumes in the following instruction cycle. For example:

```
if(eq) goto over        ; Instruction 1
r1 = 3                  ; Instruction 2
.                       ; Instruction 3
```

Instruction 2 is executed even if the condition is true and the branch is taken. If this is undesirable, a *nop* can be placed after the branch instruction, or if possible, the instructions can be rearranged. Because of this latency, a complex situation arises if successive branch instructions are coded in the following manner:

```
goto A                  ; Instruction 1
goto B                  ; Instruction 2

A:
.
B:
.
C:
```

The order of execution is instruction 1, instruction 2, A, B. If the instruction at A is not a *goto*, execution continues from B. If the instruction at A is *goto* C, the order of execution is instruction 1, instruction 2, A, B, C, and execution continues from C. Successive branch instructions are useful in some applications.

4. *Conditional Branching on DAU Conditions.* A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instruction prior to the test (a three instruction latency):

```
a0 = a0 + a1            ; Instruction 1
a2 = a0 * a2            ; Instruction 2
.                       ; Instruction 3
.                       ; Instruction 4
if(agt) goto next       ; Instruction 5
```


Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

```
.                       ; Instruction 6 (latent instruction)
```

The condition tested in instruction 5 is established by instruction 1, not instruction 2. Because of this latency effect, use the zero-latency *ifalt()*, *ifagt()*, and *ifaeq()* functions where possible (see DA Special Instructions). The DA condition tested by these conditional accumulator loads is established by the last DA instruction that affected the DAU flags.

II.VI Amiga Hardware Integration

The DSP3210 is integrated into the Amiga system as a bus-master device. It shares the system bus with the CPU. Most of what the DSP does is dependent on real-time response; it must be guaranteed a minimum response time when it requests the system bus. The DSP is given priority over the CPU when requesting the system bus. The DSP sees the entire 32-bit address range that the CPU sees, and all the caveats of programming the Amiga apply to the DSP. If a program runs awry, it can easily trash the system. The DSP can also interrupt the CPU using either a level 2 or level 6 interrupt, and the CPU can interrupt the DSP using the DSP INT1 signal. The CPU only interrupts the DSP to bootstrap it the first time the system software is loaded.

The DSP3210 can be connected to one or more I/O devices on its serial port, but only one device can be active at any given time. Both a 16-bit stereo audio CODEC and telephone CODEC are possibilities. VCOS Device drivers will be provided for both of these devices, and information on writing drivers for other devices will be available from either Commodore or AT&T.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

III. System Software

III.I Overview

A DSP is a very sophisticated piece of hardware, but it is quite useless without good software. In designing the system software for the DSP3210, AT&T set out to meet the following goals:

- *Provide maximum utilization of the silicon resource with minimum system overhead.*

  A DSP can add significant cost to a system. If it can provide only one function or one function at a time, then it becomes difficult to justify its added cost. The DSP should function well at a variety of tasks and be able to execute them simultaneously.

- *Provide a consistent and straightforward DSP programming interface.*

  DSPs can be difficult to program. To help alleviate this difficulty, standard tools and support libraries as well as a flexible model for performing I/O and communicating with DSP tasks should be provided.

- *Provide complex functions as a standard part of the system.*

  There are relatively few good DSP algorithm programmers; there are many good application programmers. This fact must be exploited in order for a DSP subsystem to be successfull. Functions for a broad range of popular DSP applications should be provided with the system.

The software support for the DSP3210 consists of two pieces: the DSP3210 Software Development Tools and the VCOS Operating System. The development tools allow application programmers to write and debug DSP3210 applications in both C and assembly language. VCOS provides a real-time multitasking environment, as well as a library of multimedia algorithms for the DSP3210. Application development involves selecting from a suite of algorithms which have already been implemented and are required by the application, or designing and implementing the DSP algorithm from scratch. Then the interface between the host application and the DSP is designed and implemented.

III.II VCOS

III.II.I Overview

VCOS provides a platform-independent environment for multitasking applications on one or more DSPs. VCOS consists of six key components:

- VCOS — The VCOS Kernel which runs on the DSP3210. VCOS runs on the DSP and has one very basic function. It traverses an execution list and executes DSP code modules in turn at a pre-determined frame rate (the default is 10ms). There are two execution lists, BLEVEL and ILEVEL (background and interrupt levels). The BLEVEL list is executed round-robin while the ILEVEL is executed once per external interrupt. VCOS also provides caching control and buffer management.

- VCAS — The VCOS Application Server. VCAS provides routines for the host system (the Amiga) to load, execute and communicate with DSP tasks.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

- VCRM[1] The VCOS Resource Manager. VCRM provides low-level resource management for the DSP. It manages the allocation of devices attached to the DSP, such as speakers, microphones, and telephone lines. It also manages run-time load balancing of tasks on multi-DSP system configurations.

- VCSIM[1] The VCOS Simulator. VCSIM provides a full non real-time simulation of the VCOS environment. VCOS tasks may be run and debugged on systems without DSP hardware.

- VCD The VCOS Debugger. VCD provides a full symbolic debugging environment for DSP applications running under VCOS. VCD supports real-time execution using both real-time sampled data and disk based data. VCD also features breakpoints and single step facilities.

- MML The VCOS Multimedia Module Library. The MML is a collection of basic and advanced signal processing algorithms implemented as VCOS modules. MML modules may be combined with other MML modules or with user created modules to create complex processing streams on the DSP. The VCOS MML consists of the following modules as of version 1.0:

  Integer sample rate converter (1->2, 1->3, 2->1, 3->1)
  16/24Kbps subband coder
  G.722 7Khz speech coder
  DTMF generator/detector
  JPEG still image coder and decoder
  MPEG level 2 audio coder and decoder

  There are other modules that will be available in the near future:

  Call progress detector [2]
  Non-integer sample rate converter
  4.7Kbps CELP coder
  Delta-Cepstrum feature extractor
  Text to phones, phones to LPC, LPC to speech
  Speaker trained, isolated word recognizer
  Speaker independent connected digit recognizer
  Talker verification
  3D graphics library[2]
  MIDI music synthesizer with EMU Proteus sound libraries[2]
  Perceptual image coder
  Perceptual audio coder
  V.22bis MNP5 Modem (V.22bis, Bell 212A, V.23, V.21, Bell 103)[2]
  V.32 modem with fallback[2]
  V.29 G3 Fax modem with fallback[2]

II.II.II Visible Caching

A VCOS application is comprised of two parts: the DSP application or algorithm and the host application. The DSP application is comprised of code "Modules" which can be combined to

1 VCRM and VCSIM are part of the VCOS 1.1 release and have not been ported to the Amiga at the time this document was created.
2 These components and modules are included in the AT&T 1.1 release

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

form a "Task". The host application communicates control and parameter information to and from the DSP task. Amiga application programmers can view and utilize these DSP tasks very much like native Amiga subtasks. Loading and preparation of DSP code is managed by the host. Code sections may be swapped in or out of on-chip memory on demand as DSP tasks are executed. There are three distinct address spaces under the VCOS DSP/Host model:

Host address	Host memory (physical, contiguous by section, locked)
DSP address	DSP physical address (mapped into host memory)
Execute address	Actual execute location (host or on-chip memory)

Modules are first loaded into host memory by the VCOS loader. A module may consist of several relocatable sections. Any or all of these sections may be cached on chip (as long as they fit in the available memory space) at the discretion of the application programmer. A module may, therefore, be represented in all three address spaces at any given time, for example:

1. Host loads module code sections into its memory (shared by DSP)
2. Host "downloads" code to DSP memory (on a bus-master just a translation or relocation)
3. Module code is cached by the DSP and is executed from on-chip memory.

In effect, all caching mechanisms are visible to the programmer, hence the term "Visible Caching". Tasks and their respective modules are executed on one of two execution lists: background or interrupt (also called BLevel and ILevel). The background list is executed in round-robin fashion (non-preemptive). The interrupt list is executed once per external interrupt and is used for time-critical I/O tasks (the code is guaranteed to be executed when an interrupt occurs). Deterministic execution is achieved through frame-based processing. Each DSP task is designed to process a block of data ("samples") in a given amount of time. The combination of visible caching and block-processing maximizes performance through optimal use of on-chip memory, and minimizes impact on host bus bandwidth by processing multiple samples per frame. Modules may save state information before exiting and load state information at execution time. Data is passed between modules and between modules and the host using "buffers". There are three types of buffers under VCOS:

AIAO	All In All Out (frame synchronous, cacheable, random access)
FIFO	First In First Out (asynchronous)
PARAM	Parameter (local shared, random access)

AIAO buffers are typically used for critical real-time DSP I/O. These buffers are static and can reside in on-chip memory (cached). AIAOs manage a fixed size data stream, and are serviced every frame. FIFO buffers are asynchronous in nature and may not get serviced every frame. FIFO buffers are used for managing sequentially accessed data streams (e.g. audio samples). PARAM buffers can contain any random access data required by the application and can be used to map host physical addresses into a DSP application. PARAM buffers have a fixed size, unlike AIAO and FIFO buffers which can have their size determined at load time. Both FIFO and PARAM buffers can be used for inter-Module and Module to Host communication. A typical application could be described by the diagram shown below in Figure 1.

VCOS also provides for special "Device Driver" Modules. These Modules are implemented to take advantage of system specific hardware (e.g., CODECs) and have standard calling interfaces. Once such a Module is written for a given device, any VCOS Module can take advantage of it for I/O. Currently, there is only one serial port, and therefore, only one active device on the DSP at any given time.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

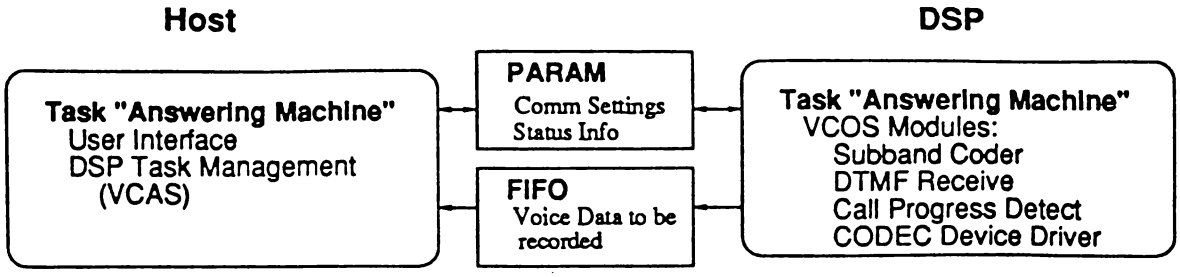


Figure 1: A Typical VCOS Application

II.II.III Sample Application

The following application loads and runs two tasks: an 8Khz device driver and a task which can feed raw 8-bit sample data to the driver. It then plays a selected data file through the task.

```
/*   play.c
 *
 *   Play Application:
 *        loads and runs iada8.tbd, then loads p_18.tbd,
 *        then, following a pause, plays out data_file thru task p_18
 */
#include <exec/types.h>
#include <exec/libraries.h>
#include <libraries/vcasbase.h>

#include <pragmas/vcas_pragmas.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/vcas_protos.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

void onbreak(void *);

struct VCASBase     *VCASBase;

#define DSPid 1
#define BUFSIZE  0x7fffL                          /* buffer size for disk i/o */

char *dd_tbd = "iada8.tbd", *app_tbd = "p_18.tbd",/* task/data file names */
     data_file[80];
int    tidDD = -1, tidPlay = -1;                  /* task id */
int    fd;
```

At this point all the required include files have been referenced: "clib/vcas_protos.h", which has all the function prototypes for VCAS as provided by AT&T, "pragmas/vcas_pragmas.h", which has all the compiler pragmas for the VCAS routines, and "libraries/vcasbase.h", which contains the structure definitions for the VCAS library base.

```
int kbhit(void)
{
     BPTR in;

     in = Input();

     return(WaitForChar(in, 1000)? 1:0);
}

void cleanexit(char *msg)
{
     if (msg != NULL) printf("Exiting: %s", msg");
     if (tidDD && (vcStopTask(tidDD) < 0))
          printf("\nvcStopTask failed\n");
     if (tidDD && (vcDeleteTask(tidDD) < 0))
```


Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

```
        printf("\nvcDeleteTask failed\n");
    if (fd >= 0) close(fd);
    if (VCASBase)
    {
        CloseLibrary((struct Library *)VCASBase);
        VCASBase = NULL;
    }
    exit(0);
}

int init(void)
{
    VCASBase = (struct VCASBase *)OpenLibrary("vcas.library", 0L);
    if (!VCASBase)
    {
        printf("couldn't open vcas.library\n");
        return (0);
    }

    if (vcAddTask(DSPid, dd_tbd, &tidDD) >= 0)
        if (vcStartTask(tidDD) >= 0)
            return (1);
        else
            printf("\nvcStartTask(%ld) failed\n", tidDD);
    else
        printf("\nvcAddTask(%s) failed\n", dd_tbd);

    return (0);
}
```

The above utility routines, *init()* and *cleanexit()* provide all that is necessary to initialize VCAS, load and start the device driver task and clean up when exiting. Note the version number for the shared library has not been determined at this time. Programming under VCAS is similiar to programming for the Amiga in that all resource allocations must have corresponding deallocations.

```
int Play(long hf, int fd)
{
    long lReadCount, lWriteCount, *lpWrite;

    if (vcGetFifoWritePtr(hf, &lWriteCount, &lpWrite) < 0)
        return -1;
    if (lWriteCount != 0x0)
        printf("... Called vcGetFifoWritePtr(), lWriteCount: %ld \n", lWriteCount);
    if (lWriteCount > BUFSIZE)
        lWriteCount= BUFSIZE; /* limit moves */
    lReadCount = read(fd, (void*)lpWrite, (unsigned int)lWriteCount);
    printf("..... read only %ld bytes from file\n",lReadCount);

    if (lReadCount < lWriteCount)
            {
        long li;
        lReadCount &= ~0x3L;    /* ensure word boundary */

        if (lReadCount == 0L)   /* wait for dsp to empty FIFO */
        {
            do
            {
                if (vcGetFifoReadCount( hf, &li) < 0) return -2;
            } while (li != 0L && !kbhit());
            if (vcUpdateFifoWritePtr( hf, lReadCount) < 0) return -3;
            printf("... Called vcUpdateFifoWritePtr() \n");
            return 0;
            }
    }
    if (lReadCount !=0)
    {
        if (vcUpdateFifoWritePtr(hf, lReadCount) < 0) return -3;
        printf("... Called vcUpdateFifoWritePtr() \n");
    }

    return 1;
}
```

The above routine reads data directly from disk into the FIFO named "fta_in". It first checks to see how much room is available in the FIFO by calling *vcGetFifoWritePtr()*. It limits the size of

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

writes to a maximum buffer size that it will be reading from disk. Once it has read the data in, it updates the FIFO indices by calling *vcUpdateFifoWritePtr()*. If it first determines that it has reached end-of-file, it waits for the FIFO to empty by checking with *vcGetFifoReadCount()*. (*Note that this busy-wait technique should actually be avoided in practice.*) It is used here for the sake of brevity.

```
main( int argc, char** argv)
{
     static long hfPlay;                              /* fifo handle */
     long   li, lj;

     (void)onbreak(cleanexit);

     if (argc!= 2) {
          printf("\nusage: play  data_file"
          "\n   data_file is the file to be played out (eg: sv.18)"
          "\nexample: play sv.18 \n");
          cleanexit(NULL);
     }

     strcpy(data_file, argv[3]);
     printf("dd_tbd: %s, app_tbd: %s\n", dd_tbd, app_tbd);

     if (init())
     {
          if (vcAddTask(DSPid, app_tbd, &tidPlay) < 0)
               cleanexit("\nvcAddTask failed\n);

          if (vcGetFifoHandle(tidPlay, "fta_in", &hfPlay) < 0)
               cleanexit("\nvcGetFifoHandle failed\n");}

          /* open the data file */
          if ((fd = open( data_file, O_BINARY |O_RDONLY)) < 0)
               cleanexit("\ncan not open data_file %s\n", data_file);

          /* preload the output FIFO */
          if (Play(hfPlay, fd) < 0)
               cleanexit("\nERROR: Play()\n");

          if (vcStartTask(tidPlay) < 0)
               cleanexit("vcStartTask failed\n");


          /* play out the file */
          for (lj = 0L; ; lj++) {
               li = Play(hfPlay, fd);

               if (li > 0) {
                  if ((lj&0x3ff) == 0) printf(".");
               }else if (li < 0) {
                    printf("\nERROR: Play() returned %ld\n", li);
                    cleanexit(NULL);
                  } else break;
          }
          /*-- done --*/

          if (vcStopTask(tidPlay) < 0)
               cleanexit("\nvcStopTask failed\n");

          if (vcDeleteTask(tidPlay) < 0)
               cleanexit("\nvcDeleteTask failed\n");
     }
     cleanexit(NULL);
}
```

The main routine parses the data file name, adds the play task and opens the data file. It then gets a pointer to the input FIFO using *vcGetFifoHandle()* and fills it with a block of data by calling the *Play()* routine once. The play task is then started - note this technique: access to the task buffers is available before the task is actually started. The file is then played out, the play task is stopped, then removed and the program exits through the *cleanexit()* routine.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

III.III Development Tools

A full suite of development tools is available for developing DSP and VCOS applications under AmigaDOS. The tools include:

d32ar	Archiver/Librarian
d32as	Assembler
d32cc	C Compiler
d32cpp	C Preprocessor
d32dump	Dump Section Information
d32ld	Link Editor
d32make	Maintain and Update Related Files
d32nm	Print Name List of Object Files
d32optim	Code Optimizer
d32sect	Relocatable Code Section Identifier
d32size	Print Section Size of Object Files
d32strip	Strip Symbol Information
d32trans	DSP32C Object Code Translator

There are two ways to develop DSP code for VCOS. One is to use the DSP3210 Assembler and the other is to use the DSP3210 C Compiler. The Assembler provides the optimum code speed and size for a DSP application. The C Compiler should only be used for non-critical code sections and for code which will not be distributed as a product. DSP applications are very time-critical. Currently, only by programming directly in assembly language can the optimum performance be achieved. The tools are organized in a directory structure as follows:

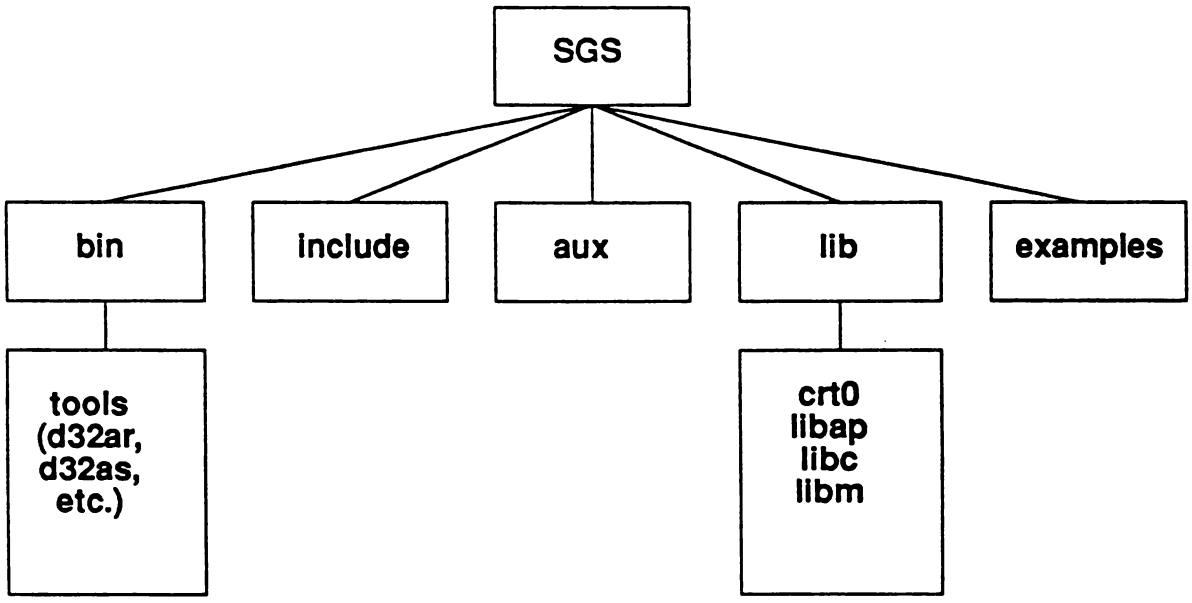


Figure 2: The DSP3210 Software Generation System (SGS)

The following environment variables are used by the tools:

DSP3210SL	The root (SGS) directory
DSP3210_AsmPP	Which preprocessor to use
DSP3210_Aux	Where the auxilliary files are

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

DSP3210_Includes	Where the include files are
DSP3210_Temp	Where to place temp files
DSP3210_Tools	Where the executable are

The auxilliary files include macros and helps files for the simulator and binary files containing boot code for the DSP3210. Refer to section V or the AT&T manuals for more information on using the tools for application development.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

IV. Software Implementation Reference

IV.I Overview

The software implementation of VCOS for the Amiga is a flexible layered architecture. It is designed to allow easy expansion or change of the underlying hardware without the need to re-implement all the higher layers of software. The reference implementation of VCOS consists of the following items:

VE.LIB	Low level hardware functions
VC.LIB	Basic VCAS functions
VD.LIB	VCAS debugger support functions
VCD.LIB	Basic VCD functions

These are all originally designed as static/linkable modules. On the Amiga, VE and VC have been abstracted into three Amiga Operating System entities: the dsp3210.resource, the dsp3210.device, and the vcas.library. The functions for VD and VCD remain static libraries, but VCD is layered onto the Amiga architecture so that it makes calls to VC through the vcas.library:

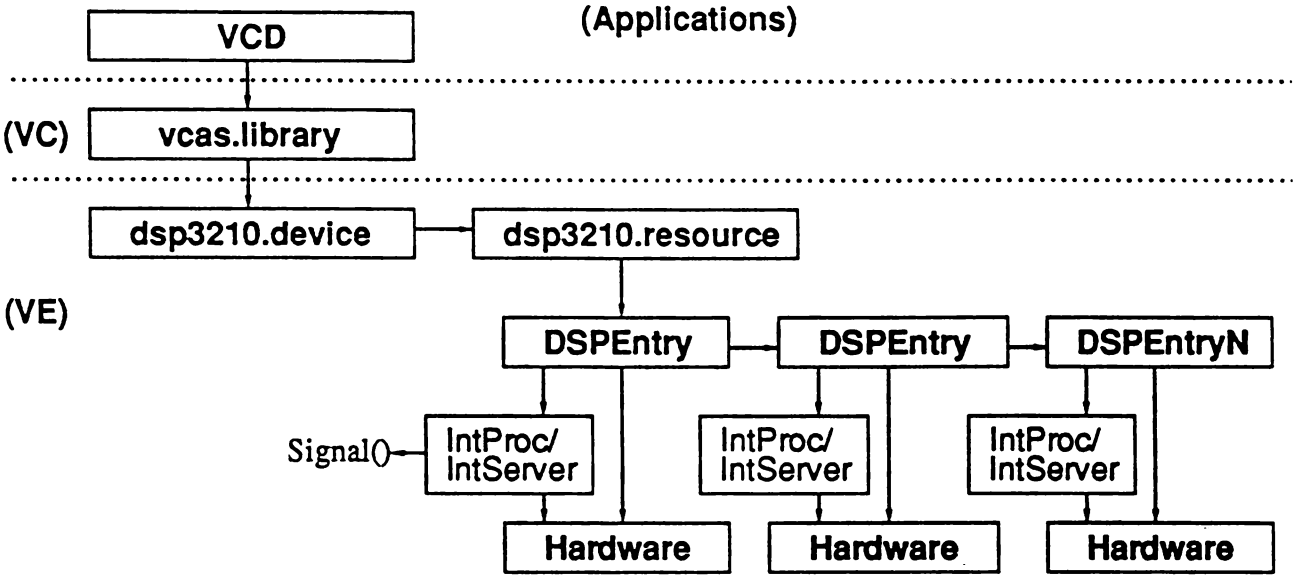


Figure 3: Amiga VCOS Software Architecture

These abstractions provide a clean and elegant system for supporting multiple DSPs as well as different hardware and software architectures in the system, including those which may be supplied by Commodore-Amiga, Inc., AT&T, or third party vendors. Most application writers will only use the VC interface of the software architecture (vcas.library). This documentation on how to access the lower layers has been provided so that people who wish to write their own DSP operating system or environment may do so in a system friendly and compatible manner.

IV.II. The dsp3210.resource

IV.II.I Overview

The resource provides hardware/vendor specific support for basic low-level functions required by

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

the system software. The dsp3210.resource appears as follows and is described in <resources/dsp.h>:

```
struct DSP3210Resource
{
     struct Node          dr_DSPNode;       /* Node to link into Exec. */
     struct MinList       dr_DSPList;       /* List of DSPs */
     UWORD                dr_NumDSP;        /* How many DSPs in system */
     UWORD                dr_Flags;         /* Special flags */
}

struct DSPEntry
{
     struct MinNode dsp_Node;               /* Link to next DSP */
     LONG (*DSPInit)();                     /* Function for Init */
     LONG (*DSPHalt)();                     /* Function for Halt */
     LONG (*DSPStart)();                    /* Function for Start */
     LONG (*DSPRdMem)();                    /* Function for Read */
     LONG (*DSPWrMem)();                    /* Function for Write */
     LONG (*DSPRegInt)();                   /* Function for Registering w/IntHandler */
     LONG (*DSPClrInt)();                   /* Function for De-Registering w/IntHandler */
     struct Task *DSPIntr;                  /* Pointer to the interrupt processor */
}
```

IV.II.II Resource Creation

A vendor will chain into the resource when their hardware is configured into the system. The vendor software should first check to see if the dsp3210.resource already exists in the system. If it does, they must chain their entries onto the existing DSP entries in the resource. If the resource does not exist, it is the vendor's responsibility to build one and then create the entries it needs. The following code fragment describes how to check for the resource and create one if it doesn't exist:

```
if (!(resource = OpenResource("dsp3210.resource")))
{
   Forbid();
   resource = AllocMem((long)sizeof(struct DSP3210Resource), MEMF_PUBLIC|MEMF_CLEAR);
   NewList((struct List *)&resource->br_DSPList);
   resource->dr_DSPNode.ln_Type = NT_RESOURCE;
   resource->dr_DSPNode.ln_Pri  = 0;
   resource->dr_NumDSP          = 0;
   resource->dr_DSPNode.ln_Name = (char *) AllocMem(RESOURCE_NAME_LENGTH, MEMF_CLEAR);
   sprintf(resource->dr_DSPNode.ln_Name, "dsp3210.resource");

   AddResource(resource);

   /* For each DSP, do the following: */
   {
      entry = AllocMem((long)sizeof(struct DSPEntry), MEMF_PUBLIC | MEMF_CLEAR);

      /* Assign function vectors here */

      AddTail((struct List *)&resource->dr_DSPList, (struct Node *) entry);
      resource->dr_NumDSP++;
   }
   Permit();
}
```

The resource remains resident in memory until the system is rebooted. In addition, the resource creates a task to manage interrupts to and from all the DSPs it is responsible for (see section II.IV below). For a motherboard based DSP, the resource will either be located in ROM or created using an AutoConfig binding mechanism like BindDrivers.

IV.II.III Resource Function Vectors

The DSPEntry functions are re-entrant and are executed via the device driver, either in the caller's

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

context if quick I/O is used, or in the device driver's context if non quick I/O is used. Their function and parameters are as follows:

```
Routine     Arguments                                              Description
DSPInit     (void);                                                Initialize all DSPs
DSPHalt     (int dspNum);                                          Halt individual DSP
DSPStart    (int dspNum);                                          Start individual DSP
DSPRdMem    (int dspNum, ULONG from, APTR to, ULONG size);         Read from DSP memory
DSPWrMem    (int dspNum, ULONG to, APTR from, ULONG size);         Write to DSP memory
DSPRegInt   (int dspNum, int modID, BYTE signal);                  Register with IntServer
DSPClrInt   (int dspNum, int Tid);                                 De-register with IntServer
```

These vectors manage all the DSPs associated with a single complete hardware entity (e.g., the motherboard, a single expansion board).

IV.II.IV Interrupt Management

The DSP3210 shares the system bus with the Amiga CPU and custom chips. The VCOS software, much like the Amiga Exec, was designed to help programmers write applications that cooperate and minimize impact on the host. Like the Amiga Exec, it is still possible to write applications which severely impact system performance, especially since the DSP is given the highest priority on the system bus. The minimal VCOS reference implementation was designed for a single-tasking system, MS-DOS. Under this implementation, synchronization between the DSP and the host occurs as follows:

> Host tells DSP to do something or requires some event to complete on the DSP
> Host busy-wait till the event condition is satisfied

So, if the host was waiting for the DSP to complete some function, it might check a PARAM buffer which was used as a synchronization flag as follows:

```
vcGetDspParamHandle(tid, "flags", param1);      /* Required for non bus-master systems */
while (!param1->flag) {{
     vcUpdateDspParam(param1);
}
```

Amiga programmers know that on a multitasking system, this style of programming can use excessive amounts of CPU time and degrade the performance of the entire system. It would be preferable to have the host application relinquish the CPU and be signalled via some interrupt mechanism when the desired condition was satisfied. A mechanism for doing this under VCOS has been provided, however, the details of the host interface differs from platform to platform. A VCOS macro called *Signal()* is provided for DSP applications to interrupt the host. The *Signal()* macro passes the VCOS Module ID and an application specific 32-bit value to the host via a FIFO.

When the resource is first created, an interrupt server is also created to manage interrupts from one or more DSPs associated with the resource. The resource maintains function vectors so that host applications may register to receive notification of an interrupt from a particular DSP (and de-register before they exit). Host applications may not interrupt the DSP at will. This is necessary so that the DSP may continue to manage real-time events without loss of data. Each DSP may have a separate host interrupt handling mechanism and associated software processor. To use the interrupt feature, an Amiga application must first register with the interrupt server. To do so, the application calls the resource function vector *DSPRegInt()*. In the future, a function vector in the vcas.library may be provided to access this function. The application passes the

Copyright â 1992, Lavitsky Computer Laboratories, Inc.

following arguments to the function:

int Dsp	Which DSP the module is running on.
int modID	The ID of the VCOS module which will cause the event.
BYTE signalnum	The signal to set when notifying the task.

Once registered, the Amiga application may use the Exec function *Wait()* to sleep until the signal is set. This brief example demonstrates proper usage of the interrupt server under VCOS:

```
vcAddTask(dspid, "sync", &tid);

signum = AllocSignal(-1);
DSPRegInt(dspid, tid, signum);

vcStartTask(tid);
signals = Wait(SIGBREAKF_CTRL_C | 1L << signum);

DSPClrInt(dspid, tid);        /* Must de-register before exiting */
FreeSignal(signum);

vcStopTask(tid);
vcDeleteTask(tid);
```

IV.III. The dsp3210.device

IV.III.I Overview

The device driver provides atomic access to low level hardware functions. It does not directly control any of the DSP hardware, but provides a standard Amiga EXEC I/O abstraction for higher level applications. All low-level abstraction is provided by the hardware resource.

IV.III.II Implementation

The device driver has functions which provide atomic access to the DSP hardware. These functions include standard Exec device management (e.g., OpenDevice()), and Exec I/O vectors which call functions in the proper vendor provided hardware resource to control the DSP. The I/O commands, flags, and errors are defined in <devices/dsp.h> and are as follows:

```
/*------ Commands ------------------------------------------------*/
#define DSP_INIT     (CMD_NONSTD)
#define DSP_HALT     (CMD_NONSTD+1)
#define DSP_START    (CMD_NONSTD+2)
#define DSP_READ     (CMD_NONSTD+3)
#define DSP_WRITE    (CMD_NONSTD+4)
#define DSP_REGINTR (CMD_NONSTD+5)

/*------ Error Returns -------------------------------------------*/
#define DSPErr_NoDSPs    1
#define DSPErr_NoUnit    2

/*------ Flags for OpenDevice ------------------------------------*/
#define DSPB_EXCLUSIVE   1
#define DSPF_EXCLUSIVE   (1<<0)
```

IV.III.III Calling The Device Driver

The device driver is opened like any standard Exec device. The only flag available to the application programmer during the open is "DSPF_EXCLUSIVE". If this flag is set, an attempt will be made to open the unit for exclusive access; if another application already has the unit open (e.g., VCAS), the open will fail. Once a unit is opened for exclusive access, all other attempts to

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

open the unit by other applications will fail until the unit is closed by the exclusive opener. The following is a brief example of how to properly open, close and perform I/O to the device:

```
// Simple example to open the dsp3210.device, do simple I/O and close
// Compile with SAS/C: lc -cfikmqw -L testdev.c
//

#include <exec/types.h>
#include <exec/memory.h>
#include <exec/io.h>
#include <devices/dsp.h>

struct MsgPort *Dport = NULL;
struct IOStdReq *Dreq = NULL;
BOOL DevOpen = FALSE;


void cleanexit(char *msg)
{
  if (msg) printf("Exiting: %s\n");
  if (DevOpen) {
   CloseDevice(Dreq);
   DevOpen = FALSE;
  }

  if (Dreq) {
   DeleteIORequest(Dreq);
   Dreq = NULL;
  }

  if (Dport) {
   DeleteMsgPort(Dport);
   Dport = NULL;
 }

  exit (0);
}

void main(int argc, char **argv)
{
  Dport = CreateMsgPort();
  if (!Dport) {
   cleanexit("init: couldn't create MsgPort");
  }

  Dreq = CreateIORequest(Dport,(long)sizeof(struct IOStdReq));
  if (!Dreq) {
   cleanexit("init: couldn't create IORequest");
  }

  if (OpenDevice("dsp3210.device", 0, Dreq, DSPF_EXCLUSIVE) != 0) {
   cleanexit("init: Error on OpenDevice()");
  }
  DevOpen = TRUE;

  Dreq->io_Command = DSP_INIT;
  Dreq->io_Flags = 0;

  DoIO(Dreq);

  if (Dreq->io_Error) {
   printf("An error occured in DSP_INIT: %ld\n", Dreq->io_Error);
  }

  cleanexit(NULL);
}
```

IV.IV. The vcas.library

IV.IV.I Overview

The VCAS (VCOS Application Server) shared library provides all high-level task management for the VCOS operating system running on the DSP3210. The philosophy behind VCAS is to

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

offload all non-time critical VCOS management from the DSP to the host. This allows the DSP to spend its valuable cycles on executing DSP functions. VCAS is defined by the MS-DOS/MS-Windows reference implementation which provides the following functions:

```
vcBootDsp(int DspN)
vcResetDsp(int DspN)
vcStartDsp(int DspN)
vcStopDsp(int DspN)

vcAddTask(int DspN, char *TaskName, int *Tid)
vcDeleteTask(int Tid)
vcStartTask(int Tid)
vcStopTask(int Tid)

vcGetFifoHandle(int Tid, Char *FifoName, long *hfHandle)
vcGetFifoReadCount(long hfHandle, long *ReadCount)
vcGetFifoWriteCount(long hfHandle, long *WriteCount)
vcReadFifo(long hfHandle, long Count, long *DestPtr)
vcWriteFifo(long hfHandle, long Count, long *SrcPtr)
vcFifoSize(long hfHandle, long FifoSize)
vcInitFifo(long hfHandle)
vcGetFifoReadPtr(long hfHandle, long *Count, long **RdPtr)
vcGetFifoWritePtr(long hfHandle, long *Count, long **WrPtr)
vcUpdateFifoWritePtr(long hfHandle, long Count)
vcUpdateFifoReadPtr(long hfHandle, long Count)

vcGetHostParamPtr(int Tid, char *ParamName, char **ParamPtr)
vcGetDspParamHandle(int Tid, char *ParamName, long *ParamHandle)
vcUpdateHostParam(long ParamHandle)
vcUpdateDspParam(long ParamHandle)
vcUpdateDspParamItem(long ParamHandle, long ByteOffset, long Count)
vcUpdateHostParamItem(long ParamHandle, long ByteOffset, long Count)
```

VCOS itself is a small, tightly coded kernel running on the DSP. DSP application programmers make use of the following macros to implement their applications and communicate with the host system:

```
NewModule(EntryName)
NewSection(SectionName, SectionType)
AppendSection(SectionName)
Push(rS)
Pop(rD)
GetSectionSize(SectionName)
SetEntryPoint(EntryPoint)

LoadSection(SectionName)
SaveSection(SectionName)
LoadRunSection(SectionName, EntryPoint)

WriteFifo(FifoSectionName, ByteCount, SourceAddress)
ReadFifo(FifoSectionName, ByteCount, DestinationAddress)
GetFifoSize(FifoSectionName)
GetFifoWriteCount(FifoSectionName)
GetFifoReadCount(FifoSectionName)

AddressPR(Label)
SetBaseSR(Label, rB)
SetBaseHSR(Label, rB)
AddressSR(Label, rB)

Signal(Value)
```

Documentation for all of the above functions may be found in the VCOS Technical Reference (or refer to section V for a DSP programming example). There are several special files which are created to define the structure of each VCOS application which the shared library must be able to read (specifically during AddTask). These are the VCOS Task Build Files (TBDs), and the VCOS Loader Files (LDFs). The VCOS application programmer must be aware of these file formats and know how to manipulate them. In the future, there may be calls added to VCAS which abstract the file formats into VCAS system calls.

Copyright © 1992, Lavitsky Computer Laboratories, Inc.

IV.IV.II Implementation

Application programmers open and make calls directly to the VCAS library. When the library is first initialized, it determines how many DSPs are available in the system (using the dsp3210.resource). It then attempts to open each DSP for exclusive access (using dsp3210.device). Once all the DSPs have been allocated, the library initializes VCOS for all available DSPs. Currently, the library uses the default VCAS mechanism of searching the VCOS.CFG file to determine how many DSPs there are in the system rather than using the provided resource. The need for this file may be removed in a future version of the software.

IV.IV.III Calling The VCAS Shared Library

The VCAS shared library is opened like any standard Exec library. The library base symbol for the C compiler is "VCASBase". The following is a brief example of opening the VCAS shared library and making calls to VCAS:

```
// Simple example to open the vcas.library, call VCAS and close
// Compile with SAS/C: lc -cfikmqw -L testlib.c
//
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/io.h>
#include <libraries/vcasbase.h>
#include <pragmas/vcas_pragmas.h>

main()
{
  int tidl;

  if (VCASBase = OpenLibrary("vcas.library", 0L) == NULL) {
   printf("Couldn't open vcas.library\n");
   exit(0);
  }

  if (vcAddTask(1, "ata", &tidl)) {   // Now make a VCAS function call
            printf("vcAddTask() failed\n");
   } else
            /* Additional VCAS calls may go here */{
            vcDeleteTask(tidl);
   }

  if (VCASBase) CloseLibrary(VCASBase);
}
```

Refer to the VCOS Technical Reference Manual for information regarding .TBD and .LDF files for DSP Tasks.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

V. Application Development

V.I Overview

While a C Compiler does exist for the DSP3210, the DSP3210 Assembler is the most efficent and reliable tool for developing VCOS applications. The Assembler is the only route to producing the most optimal code (both in performance and size), which can often be a critical issue in a multitasking DSP environment.

V.II Using The Tools

The following brief example will serve to demonstrate the basic requirements for writing a DSP application:

```
#include <macros.h>

NewModule(main3)
NewSection(main3, sPROG)
NewSection(cmain3, sPROG)
NewSection(piparm, sPARAM)

AppendSection(main3)
     *sp++ = r18

     LoadSection(cmain3)       /* Cache these sections */
     r3 = cmain3
     call r3(r18)              /* Run our cached code */
     nop

     sp = sp--
     r18 = *sp
     nop
     return(r18)
     nop

/* Here begins the actual main program */
AppendSection(cmain3)
emain:
     *sp++ = r18               /* Push r18 onto stack */

     !r4 = (long)var1
     r3 = (long)*r4
     r3 = (long)r3 + 1
     *r4 = (long)r3

exitemain:
     sp = sp--                 /* Decrement stack pointer */
     r18 = *sp                 /* Pull r18 off stack */

     return(r18)
     nop

/* A PARAM buffer for exhanging information with the host */
AppendSection(piparm)
var1:     long      1          /* Param variable 1 */
var2:     long      2          /* Param variable 2 */
```

To assemble the above example, use the following command from the CLI/Shell:

```
d32as -b big -l -i INCLUDE:dsp -o test.o test.s
```

`-b big` Directs the assembler to generate code for a DSP3210 i n big-endian mode.
`-l` Directs the assembler to generate a full listing file.
`-i` Directs the assembler where to locate include files.
`-o` Directs the assembler where to place the resulting object file.

Copyright ã 1992, Lavitsky Computer Laboratories, Inc.

After assembling, the application must be linked using the linker as follows:

```
d32ld -map -r -o test test.o
```

`-map`	Directs the linker to generate a memory map to standard output.
`-r`	Directs the linker to keep relocation information in the resulting object file.
`-o`	Directs the linker where to place the resulting object file.

The VCOS application is then ready for loading using the VCAS *vcAddTask()* call, VCD, or VCSIM.

V.III Debugging techniques

Both VCD and VCSIM can be powerful tools for debugging VCOS applications. VCD can be used to step through applications running in real-time on the DSP. In VCD breakpoints can be set, registers and buffers examined, and code stepped through. With VCSIM, a full DSP3210 and VCOS environment are simulated in software. VCSIM combines the simulation capabilities of d32sim with the debugging features of VCD. This makes it possible to develop and debug applications without having DSP hardware available (albeit much more slowly). VCSIM also removes the hardware variable from the debugging equation, which means faulty hardware is less likely to be the root cause of a problem.

Aside from using VCD and VCSIM, there are other useful methods of debugging applications in real-time. A VCOS application is typically a "black box". It is difficult at best for a programmer to know what is really happening inside his application while it is executing. One good technique (especially on bus-master systems like the Amiga) for debugging a VCOS application is to use debugging variables or counters in PARAM or FIFO buffers within the DSP code. A lot of time and trouble can also be saved by first building an external reference model for the application in C or C++. It is much easier to try new ideas in the higher level language than to laboriously code changes or new routines in assembler which may not even work, let alone be used in the final application.

Copyright â 1992, Lavitsky Computer Laboratories, Inc.

VI. References

1. "DSP3210 Digital Signal Processor. The Multimedia Solution", Information Manual. AT&T, September 1991 printing.

2. "VCOS Multimedia Development Kit", Technical Reference". AT&T Release 1.0, March 1992 printing.

3. DSP3210 Support Software Toolkit Manual, Release 1.3

4. DSP3210 Support Software Library Manual, Release 1.3

VII. Acknowledgements

The author would like to thank the following people for their help and invaluable contributions to this project:

From Commodore:
Jeff Porter
Randell Jessup
Dave Haynie
Debashis Pramanik

From AT&T:
Gary Murakami
John Lynch
Charlie Mera

Copyright â 1992, Lavitsky Computer Laboratories, Inc.

Amiga Developers Conference Notes
Orlando, 1993

Copyright

Copyright © 1993 Commodore-Amiga. Inc. All rights reserved. The materials used herein have been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions. Commodore and Amiga are registered trademarks of Commodore Electronics Ltd. and Commodore-Amiga, Inc. respectively. This document may also contain reference to other trademarks and registered trademarks for the various products listed which are believed to belong to the sources associated therewith.

Warning

The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

Disclaimer

This information is provided "as is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any claim arising out of the information presented herein, even if it has been advised of the possibility of such damages.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks.

Amiga is a registered trademark of Commodore-Amiga, Inc.

Amiga 600, Amiga 1200, Amiga 4000, AmigaDOS, Amiga Workbench, and Amiga Kickstart are trademarks of Commodore-Amiga, Inc.

Commodore Dynamic Total Vision and CDTV are registered trademarks of Commodore Electronics Limited and Commodore.

AmigaVision is a registered trademark of Commodore Electronics Limited and Commodore.

Amiga/NFS, CBM, Commodore, the Commodore logo, and AUTOCONFIG are registered trademarks of Commodore Electronics Limited.

dBase is a registered trademark of Ashton-Tate Corporation.

68000, 68020, 68030, 68040 and Motorola are trademarks of Motorola, Inc.

Aztec C and Manx are trademarks of Manx Software Systems.

SAS and Lattice are registered trademarks of SAS Institute, Inc.

UNIX is a registered trademark of AT&T.

Novell and Netware are registered trademarks of Novell, Inc.

IBM is a trademark of International Business Machines, Inc.

Macintosh is a trademark licensed to Apple Computer, Inc.
Apple is a trademark of Apple Computer, Inc.

MS-DOS is a trademark of Microsoft Corporation.

V39 and AA Compatibility

by Commodore Engineering and CATS

Introduction

The capabilities of the built-in Amiga graphics hardware did not change significantly between the introduction of the Amiga 1000 in 1985 and the release of the Amiga 3000 in 1990. The ECS chipset and the display enhancer added several new graphics modes and increased the functionality of existing modes -- yet the market demanded more.

This document explains some of the features of the latest generation Amiga graphics hardware known as the AA (double-A) chipset, and ways to ensure application compatibility with these and other V39 features and changes. Additionally, coding methods are outlined which will allow current software to automatically exploit some of the new features.

New Hardware Features

The most important capabilities of the new Amiga graphics hardware are summarized below.

- Enhanced Bandwidth. A 32-bit wide data bus allows doubling of Chip memory bandwidth (up to 2x the normal bandwidth) and supports the input of 32-bit wide bitplane and sprite data. Bandwidth may be doubled again (to 4x) by using Fast Page Mode RAM.
- More Bitplanes. The maximum number of bitplanes has increased to 8 in all resolution modes. This translates to a 256-entry color table for each available mode.
- Larger Palette. Each entry in the color table may now be 25 bits wide (8 bits each for Red, Blue, and Green data plus 1 bit for genlock information). This translates to a palette of 16,777,216 colors.
- HAM and HALFBRITE are available in all display resolutions and depths, allowing for greatly enhanced display modes such as 8-bitplane HIRES HAM.
- Enhanced Dual Playfield Support. Each playfield may now have up to 4 bitplanes. The bank of 16 colors in the 256-color table is independently selectable for each playfield.
- Enhanced hardware scrolling support. The resolution of bitplane scrolling has increased to 35ns.

- Enhanced Sprite Support. Sprite resolution can be set to Lores, Hires, or SuperHires, independent of screen resolution. Attached sprites are now available in all modes. However, some new higher bandwidth modes may only allow one sprite (making an attached sprite impossible). Odd and even sprites may use their own independent 16-color bank from the 256-color table. Old format sprites are still 16 bits wide, but new format sprites may be 32 or 64 bits wide. Sprites may now optionally appear in the border region. The horizontal positioning resolution of sprites has increased to 35ns (equivalent to SuperHires pixel widths.)
- Hardware scan-doubling support. 15 kHz bitplanes and sprites may now be scan-doubled for flicker-free display on 31 kHz monitors, and for enhanced display sharing with 31 kHz bitplanes.
- ECS compatibility. New chips will power-up in an ECS compatibility mode, which will allow many older self-booting programs to be run on new machines.

Ensuring Compatibility

This section covers programming techniques that will help ensure that your application will work as expected when running under V39 and on systems that use AA and future generation graphics hardware.

Support of Release 2 is integral to supporting new graphics chips. One of the main reasons is the display database. Starting with the ECS chipset, not all machines have all display modes available. This will be even more true in the future. The only way to know if a particular mode is available is to check the display database.

For a detailed explanation of how to properly open a screen using the display database information to check for available modes, refer to the *Amiga ROM Kernel Reference Manual: Libraries*, 3rd Edition. Also see the Amiga Mail articles entitled "An Introduction to V36 Screens and Windows" (page IV-3) and "Opening Screens and Windows on Any Amiga" (Page IV-17). Some other pertinent details can also be found in the Paris DevCon notes article entitled "Monitors, Modes, and the Display Database."

Once a screen or other display has been opened, all manipulations should also be handled using system calls. In particular, these manipulations must be handled by the system software:

- Allocation of graphics resources - use the graphics or intuition library
- Palette manipulation - use the graphics library
- Manipulating icons - use the icon library

- Manipulating mouse pointer sprite - use intuition.library to select or change imagery; input.device to move it
- Manipulating sprites other than the mouse pointer - use the graphics library
- Allocating and changing colormaps and colortables - use the graphics library
- Bitplane allocations - use AllocBitMap() under V39 and higher, and AllocRaster() under earlier releases, but please refer to the next section for important information about limitations of AllocRaster().

Furthermore, these manipulations should be handled by the system software whenever possible:

- Drawing operations - use the graphics library and Intuition library
- Blitter operations - if possible, use only use higher level blitter functions from graphics.library such as BltBitMap(), BltRastPort(), etc. If you must wait for a blit to complete, always use the system's WaitBlit() function, never your own. The system knows about and handles problems with the BLITTER_DONE signal that different revisions of the Agnus chip have. If you feel that must use the blitter hardware directly (sacrificing all hope for any RTG compatibility), be sure to properly OwnBlitter, then WaitBlit, then use blitter, then DisownBlitter.

Remember that system structures are subject to change or extension. For new graphics hardware, structures likely to be extended are those associated with ColorMaps, ViewPortExtras, Sprites, and many others. To deal with changing system structures, developers should use system calls to allocate, create, and manipulate these structures. Many new "Get" and "Set" calls have been added to graphics.library to provide an upwards-compatible mechanism for reading and setting structure members which were previously accessed by program code either directly or through macros which accessed them directly. For future compatibility, do not directly access or change any structure or value for which a "Get" and "Set" call is available. And do not declare or AllocMem any structure for which a special allocation call is provided.

Most of the new "Get" and "Set" graphics calls for V39 are listed below. Note that some of these functions are workalike replacements for old gfxmacros which poked or peeked a structure, and others provide tag-based capability for getting and/or setting more than one attribute (Attr) of a structure:

```
SetOutlinePen(rp,pen)(a0,d0)
GetBitMapAttr(bm,attrnum)(a0,d1)
SetRPAttrsA(rp,tags)(a0/a1)
GetAPen(rp)(a0)
GetDrMd(rp)(a0)
SetABPenDrMd(rp,apen,bpen,drawmode)(a1,d0/d1/d2)
SetMaxPen(rp,maxpen)(a0,d0)
SetWriteMask(rp,msk)(a0,d0)
GetRPAttrsA(rp,tags)(a0/a1)
GetBPen(rp)(a0)
GetOutlinePen(rp)(a0)
```

Use the newest functions available. For compatibility reasons, it is sometimes not possible to change the behavior of an existing system function to provide enhanced capabilities. In such cases, new functions may be provided. For upwards compatibility, conditionally use the newest functions that are available. For example, when running under an OS earlier than V39, applications that wish to allocate a BitMap should use AllocRaster() and InitBitMap(). New or updated applications should use the new AllocBitMap() call instead when running under V39 or higher. See the V39 graphics library Autodocs for specifications of new functions such as AllocBitMap().

Hardware Compatibility

The new chips power up in an ECS compatible state that allows a large portion of older, self-booting (i.e., game) software to run even though these older titles often access hardware registers directly. However, directly accessing hardware should be avoided by any software that expects to run after the operating system has been started. Keep in mind also that it is difficult to maintain hardware register-level compatibility even on powerup as new and more enhanced chipsets are developed. Avoid directly programming the hardware whenever possible.

In cases where applications *must* access the hardware:

- Applications must not write spurious data to, or interpret data from, currently unused bits or addresses.
- Applications must set undefined bits to zero for writes and ignore them for reads.
- Mask out all the bits except the ones the application is actually interested in.
- Do not assume the initial state of any registers.
- Do not read write-only registers or write to read-only registers.
- Do not read or write *bytes* to the custom chip registers (they are *word* registers).

Graphics and Intuition Issues

The following notes detail some known coding practices that can cause software to function incorrectly on V39 and AA machines.

- BitMap plane allocations unsuitable for new modes. This problem surfaces because the new chips have an increased fetch bandwidth. The data for each bitplane scan line must be properly aligned in Chip memory for new display modes to work.

For instance, these allocation methods should all be avoided under V39 and higher:

```
/* WRONG for AA */
for(planenum=0; planenum<DEPTH; planenum++)
   bitmap.Planes[planenum]=(PLANEPTR)AllocMem(RASSIZE(width,height),MEMF_CHIP);

/* Also WRONG */
allplanes=AllocMem(DEPTH*RASSIZE(width,height),MEMF_CHIP);

/* Also WRONG */
for(planenum=0; planenum<DEPTH; planenum++)
   bitmap.Planes[planenum]=(PLANEPTR)AllocRaster(width,height);
```

The correct V39 method for allocating rasters is to use AllocBitMap() for graphics-level bitmaps and for CUSTOMBITMAP screens, or let Intuition OpenScreen() or OpenScreenTagList() allocate your bitmap. These methods will allow you to get the greater bandwidth and more efficient displays available under V39 in a manner that will be upward-compatible with future enhancements. The new V39 graphics function AllocBitMap() properly handles all allocation and alignment issues. The Intuition OpenScreenTags() call uses AllocBitMap() under V39.

If for some reason your existing code design prevents you from conditionally calling AllocBitMap() or OpenScreen() when running under V39 and higher, you may be able to at least get compatibility with the currently available higher-bandwidth modes by using AllocRaster() or AllocMem() after rounding your width up to a multiple of 64 pixels (absolutely no guarantees here; future chips may require greater alignment; we suggest you conditionally code to use AllocBitMap() for future compatibility).

- Incorrect assumptions about BitMap->BytesPerRow. Applications that use system functions to initialize and allocate BitMaps and their elements should be aware that the value of BitMap->BytesPerRow may be different from the expected value, depending on the bandwidth mode chosen, the custom chip type, method of allocation, and the asked-for width of the allocated planes.

Here's the explanation. First, due to fetch-alignment restrictions in AA, some modes require a higher granularity (BytesPerRow must be a multiple of 4 or 8, instead of just 2 under ECS). Second, BytesPerRow actually means two different things, which have always been identical, but aren't any more when using interleaved bitmaps. BytesPerRow officially means "the number of bytes you have to add to a pointer to a byte of the BitMap to get to the same place one row down." It no longer can be depended on to mean "the number of bytes in this row."

To detect software compatibility problems related to BytesPerRow changes, it is extremely important to test on a machine with the AA chipset and Mode Promotion turned on in the IControl Preferences editor, and if the software supports various display sizes, to test with sizes which are not divisible by 64. Two typical symptoms of BytesPerRow problems are 1. skewing of the display, where the pixel data for every line is progressively further right or left, giving a diagonal appearance to the display, and 2. images saved with excess blank space at the right.

> *Note* - CATS intends to provide a developer tool, probably called "BumpBPR", which will force all Screen and AllocBitMap() BitMap widths to a multiple of 64 pixels to emulate the higher scanline alignment of higher bandwidth AA modes. This should allow developers to test for many BitMap->BytesPerRow problems on non-AA and no-promotion machines running V39.

- Non-matching BitMap->BytesPerRow Some applications assume that two BitMaps of the same width, but allocated by different methods, will be swappable or block-copyable in various manners. This is often no longer the case. For example, a BitMap allocated by OpenScreen (which calls AllocBitMap) may have raster line storage widths rounded up to provide higher alignment for higher-bandwidth displays. But the raster lines of BitMap planes allocated with AllocMem() are generally rounded up only to a multiple of 16 with a hardcoded macro (RASSIZE), and for compatibility reasons, AllocRaster() currently still performs only the same rounding to a multiple of 16.
- Interleaved BitMaps For interleaved bitmaps, BytesPerRow is quite a bit larger than the number of bytes in this row. Screen grabbers and screen printers seem to be the primary victims of this change. For compatibility, the Workbench screen is non-interleaved if it opens before IPrefs has run. If opened or reset after IPrefs has run, the Workbench screen will be interleaved, so under V39, the Workbench screen in a normally booted environment is likely to be interleaved. GetBitMapAttr(bm, BMA_WIDTH) can return the true width of a BitMap in pixels. However, this is not the width to be used when saving a BitMap as an ILBM since this width may be rounded up for alignment reasons.
- Incorrect assumptions about the internal structure of BitMap plane data. New types of BitMaps may have a significantly different layout. For example, new V39 interleaved BitMaps are allocated as one contiguous block of Chip memory and are internally different from non-interleaved plane data.

Interleaved BitMaps consist of line 0 of plane 0, followed by line 0 of plane 1, line 0 of plane 2, on through to line 0 of plane n. Then the pattern repeats with the data for the next line. This BitMap layout reduces the color flashing effect which normally accompanies the blitting of individual planes. Interleaved BitMaps can be requested by applications under AA. Note however, that callers are not guaranteed to receive an interleaved BitMap whenever they ask for one. The BMF_INTERLEAVED flag to AllocBitMap() is considered a request, not a requirement. If no sufficiently large continuous block of chip memory is available, it may not be possible to allocate the BitMap interleaved. In that case, AllocBitMap() will attempt to allocate an ordinary BitMap instead. Applications should be written so as not to be sensitive to whether or not a BitMap is interleaved. In those rare cases when it might matter, GetBitMapAttr(BMA_FLAGS) can be used.

- Incorrect assumptions about the Display Database. Data about a display mode could change between one version of Kickstart and another, between different models of the Amiga and even between similar models. Treat the data in the display database as dynamic! Information that was available under V37 may not necessarily be available under V39. For instance, VGA and A2024 mode information is no longer in ROM and unless they are added by the user, there will be no information about these modes available.

Never cache information about the DEFAULT_MONITOR_ID modes because the default monitor can be changed by the user with the promotion feature.

Use the 2.1 asl.library screen mode requester to present your user with choices which are actually available (Note - 2.1 asl.library works under 2.0 Kickstart, and a free amendment to distribute 2.1 asl.library is available for 2.0 Workbench licensees).

- Direct handling of ViewPorts. Applications that use low-level graphics calls to create the View directly may have problems in V39. MakeVPort() and MrgCop() can now return an error. In addition, the gap required between ViewPorts may now need to be significantly larger than on pre-AA chips, where it was never more than 3 non-interlaced lines (6 interlaced lines). The V39 graphics function CalcIVG() (Calc Inter Viewport Gap) may be used to determine how many blank lines will be needed above a ViewPort. Note that Intuition currently uses at least 3 lines, or MAX(3,CalcIVG(v,vp)) (substitute 6 for 3 if interlaced). Also note that for old display modes with old depths and 4-bit-per-gun colors, the gap is unchanged from that under the ECS and original chipsets.

- Incorrect use of PAL or NTSC flags. The common method of determining whether a machine is running in PAL or NTSC mode has always been to check for the PAL bit in GfxBase->DisplayFlags:

```
BOOL IsPAL;

IsPAL=(GfxBase->DisplayFlags & PAL) ? TRUE : FALSE;
```

One classic reason for desiring such a PAL/NTSC determination is to decide what size or type display to open. The other common reason is to determine what clock constant to use in serial or audio period calculations (Amigas built as PAL machines have a slightly different system clock crystal frequency than Amigas built as NTSC machines, and serial/audio period calculations are dependent on this frequency).

Since V37 and V39 are more adaptable, this bit may no longer be counted on to mean the user's preferred display mode, nor to even signify the probable clock crystal in the user's system.

For accurate serial and audio period calculations, you need to determine whether the system has a system clock crystal designed for PAL or NTSC. There is no totally foolproof way of doing this. Under 1.x through 2.x, your best indicator of whether a system has a PAL system clock crystal is the GfxBase->DisplayFlags PAL bit. This bit should tell you the hardware default of the machine, and should only mislead you if the user has moved the PAL/NTSC jumper or has run a PD program to change the bootup mode between PAL and NTSC.

Under V39 (and probably above), new BootMenu options now allow the user to software-select PAL or NTSC as the default graphics environment. This can modify the GfxBase->DisplayFlags PAL bit, but obviously does not change the system clock crystal. Therefore, under V39 and higher, a new DisplayFlags bit called REALLY_PAL has been added to signify the whether the motherboard hardware jumper setting appears to be PAL. Therefore, under V39 and higher, the best bit available for determining NTSC or PAL hardware is the new REALLY_PAL bit.

If your software needs to determine the characteristics of default displays, prior to V37 look at DisplayFlags PAL bit. Under V37 or higher, instead use GetDisplayInfoData() to get information about the modeid LORES_KEY or HIRES_KEY, and then check the DisplayInfo.PropertyFlags for properties such as DIPF_IS_PAL (which will be true for both PAL and DblPAL).

If your software instead needs to determine the characteristics of the user's preferred display mode (as evidenced by the mode of her Workbench or other default public screen), you can GetDisplayInfoData() on the modeID of the default public screen. Be aware that this modeID *might not* be any of the PAL or NTSC modes, but rather VGA or some other mode.

```
#include <graphics/displayinfo.h>

Bool IsPAL;
struct Screen *screen;
ULONG modeID = LORES_KEY;
struct DisplayInfo displayinfo;

/* Open graphics.library, etc. */

/* Above, modeID is initialized to LORES_KEY.  You should inquire about LORES_KEY
 * or HIRES_KEY if you are interested in the display characteristics of default
 * displays (for example, what OpenScreen() will provide if you don't use tags to
 * ask for a display mode with an explicit monitor like PAL or NTSC or VGA).
 *
 * Do the following LockPubScreen part if you are instead interested in the display
 * characteristics of the modeID of the user's default public screen (usually
 * Workbench).
 */

if (screen = LockPubScreen(NULL))
    {
    modeID =GetVPModeID(&(screen->ViewPort));
    UnlockPubScreen(NULL,screen);
    }

/*
 * Now get information about the modeID
 */
if (GetDisplayInfoData(NULL,(UBYTE *)&displayinfo,sizeof(struct DisplayInfo),
                                        DTAG_DISP, modeID))
    {
        /* True if PAL or DblPAL */
    if (displayinfo.PropertyFlags & DIPF_IS_PAL)
        IsPAL = TRUE;
    else
        IsPAL = FALSE;
    }
```

- Incorrect assumptions about ColorMaps and ColorTables. The color system has undergone significant changes so new V39 graphic functions should be used to manage color whenever possible. ColorMaps should always be allocated using GetColorMap(); freed using FreeColorMap(); colors changed using LoadRGB4/32(), SetRGB4/32(), or SetRGB4/32CM(); and colors queried using GetRGB4/32(). Specifically, the value ColorMap->ColorTable and the structure it points to should never be poked or read directly.

The color functions with names containing "32" are new V39 functions for getting, setting, and loading color registers. These new functions treat color guns (R, G, B) each as 32-bit values for handling not only the 8-bit color available in AA but also any conceivable future needs. Use of these new 32-bit color functions is required to display the 16 million different color shades available under AA and V39. The old 4-bit color manipulation functions can only provide 4096 different colors. Use these new 32-bit color functions to ensure the future compatibility of your V39 applications.

When using 4 or 8-bit R/G/B values, scale your values to 32 bits. Scale 8 bit R/G/B values to 32 bits by duplicating the 8-bit value in all 4 bytes of the 32-bit value. Scale 4-bit values to 32 bits by duplicating the 4 bit value in each nibble of the 32-bit value.

4-bit red value $3 becomes $33333333
8-bit red value $1F becomes $1F1F1F1F
8-bit red value $03 becomes $03030303

The graphics.library VideoControl() function should be used to get, set, or clear the special features associated with ColorMaps.

- Poking Copper lists or copinit. The structure and order of Copper lists will change for all modes, so any program that relies on poking the Copper lists will likely break. Certain programs (especially games) make assumptions about copinit and poke it directly. These programs will break with the AA chips and V39 unless the AA machine is running in a non-AA old-chip emulation mode. Note again that copinit has changed, and will continue to change. Do Not Touch.
- Dangerous bits in the display hardware. Many bits in the custom chip registers that were previously undefined now have a function. Beware of poking the following bits.
- In BPLCON0: bits 0, 4, 5, 6, and 7
- In BPLCON2: bits 7, 8, 9
- In BPLCON3: bits 0,1,6,7, and bits 9 through 15
- Illegal use of the processor to write to bitplane and sprite data registers. Bitplane and sprite data registers should NOT be written to by the processor. The correct way for data to be fed to the chips is by setting up DMA that points to the data in Chip memory.
- Illegal re-use of sprites on the same line. Attempting to re-use sprites on the same horizontal line by redefining the sprite data is illegal and unsupported. Sprites may only be re-used vertically, and then only with at least a one line gap between re-uses. See the *Amiga Hardware Reference Manual* for an example of sprite re-use.
- Incorrect allocation of sprite image data. Because of the increased fetch bandwidth of the new chips, sprite image data must be properly aligned in Chip memory. Under 2.0, the best way to allocate this memory is to call the Exec library AllocMem() function. Use the new AllocSpriteData() call for allocation of new mode sprites under V39 and higher.
- Using an attached sprite for the Intuition pointer was quasi-supported under 2.0. It no longer works.

- Forgotten FreeSprite() call. Sprites may now have a number of different modes. However, under Intuition, these modes are global to all sprites. If a program gets a sprite in a particular mode, but then does not free it, Intuition (and all Intuition-based programs including Workbench) are forced to use sprites in the mode of the forgotten sprite. In general, it is good practice to not use sprites other than the pointer when coding software that is meant to be Intuition-compatible.
- Illegal use of Intuition pointer data. Intuition's pointer handling has become much more sophisticated. People playing tricky games with the pointer data may get into trouble if they make assumptions about the data format of the Intuition pointer which can change depending on the display mode.
- The old SetPointer() and ClearPointer() functions still work, giving compatible Intuition pointers. Likewise, the pointer imagery in devs:system-configuration will be used until a pointer.prefs file is received. However, due to growing complexity in the pointer subsystem, calling SetPointer() or ClearPointer() from within an input handler or inside Begin/EndRefresh() runs a risk of deadlocking. There are currently some patches in place to help prevent a deadlock, however we warn people to stick to simple rendering functions only when inside LockLayer(), LockLayerInfo(), BeginRefresh(), BeginUpdate(), etc. and not to call Intuition or other high-level system functions inside of LockIBase(). As well, great care should be taken inside an input handler (it's generally best for the handler to signal a high-priority task which actually does the work). Don't be the application that dies under the next release when an inappropriate function that happened to work now deadlocks because its handling has become more sophisticated. See the Autodocs for these functions for more information on what other system calls are OK to use with these functions.
- The pointer information returned by GetPrefs() is no longer kept up-to-date, since the pointer data can exceed the storage space available in struct Preferences. (The ROM default pointer will be returned in all cases). (Like V37, V39 ignores the pointer data in calls to SetPointer() after the first one, for reasons such as this).
- OpenScreen failure if too deep. Since all modes and all depths are no longer available on every systems, OpenScreen will fail if you request a mode/depth combination that is not displayable. This can be an issue both for applications that may have been counting on allocating extra planes this way, or counting on having extra planes that would not be displayed. There is a new SA_ErrorCode value, OSERR_TOODEEP.

- Intuition and Graphics are involved in a scheme to maximize compatibility with older sprite-using applications. The AA chipset supports variable sprite resolution and width, but the setting affects all sprites. If an application requests a specific sprite width (through the old graphics.library/GetSprite() call or the new graphics.library calls (GetExtSpriteA() and ChangeExtSpriteA()), and Intuition's sprite is not compatible with that request, then graphics.library will blank Intuition's sprite and notify Intuition. Intuition will rebound by generating the most suitable pointer sprite which is compatible.
- Pens for Screens. The handling of defaults for the pen array is a bit involved because of compatibility issues. The only change for old applications should be those who pass an SA_Pens array of {~0}. They will get the new values for BARDETAILPEN, BARBLOCKPEN, and BARTRIMPEN. This will only affect the color of their screen title bar, and the color of the menus of any window on that screen which requests new-look menus (WFLG_NEWLOOKMENUS). Applications that specify no SA_Pens array or ones who specify an SA_Pens array with at least one explicit pen provided get V37-compatible defaults. Our options were limited because the request for default palette and default pens were too loosely coupled (until SA_LikeWorkbench).
- Screen.BitMap is becoming obsolete. The original Screen structure has an embedded instance of a BitMap structure, which can not grow to support current needs. When Intuition allocates a on-CUSTOMBITMAP screen's BitMap, it now uses AllocBitMap(), and just copies the old struct BitMap portion into the embedded `&screen->BitMap`. Intuition internally now references the true BitMap (obtainable as `screen->RastPort.BitMap`) instead of `&screen->BitMap`. We recommend that applications do the same. This is the direction we're headed anyway for RTG, and is needed for double-buffering. There are currently some patches in Intuition to handle people who are poking the `Screen.BitMap` depth or planes. Such poking is strongly discouraged. See the new V39 Intuition function ChangeScreenBuffer().
- Console compatibility Applications which draw graphics in the console portion of the window, but use console scroll commands may not work if they are drawing their text in other than pen 1 or pen 0 when only the bitplane for those colors is scrolled.

Applications that use pen sharing on their own screens should allocate and set pens 3-7 if they plan on using those colors in console. The console device still uses pens 0-7 just as it always has.

Promotion-Related Issues

Under V39 and AA, when Mode Promotion is turned on, display modes which use the default monitor (e.g., display modes that are not explicitly PAL or NTSC) are promoted to scan-doubled or de-interlaced modes such as DblPAL or DblNTSC to provide a flicker-free display. Mode Promotion, on or off, is controlled system-wide by the IControl Preferences editor and the availability of the DblNTSC or DblPAL monitor.

Mode promotion can change the behavior of the system in a manner which may be incompatible with certain applications.

- The overscan limits of DblNTSC (DblPAL) are a little less than the overscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to, but still less than, NTSC (PAL)).
- It may be harder to center a DblNTSC or DblPAL screen on certain multiscan monitors than it was to center a hardware de-interlaced NTSC or PAL screen. (3.01 provides additional centering flexibility which helps solve this problem).
- An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists and copper-list pokers.
- The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignmentoverscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to, but still less than, NTSC (PAL)).
- It may be harder to center a DblNTSC or DblPAL screen on certain multiscan monitors than it was to center a hardware de-interlaced NTSC or PAL screen. (3.01 provides additional centering flexibility which helps solve this problem).
- An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists and copper-list pokers.
- The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignment currently required by AA). However, if the custom BitMap is an unusual width, it may not be sufficiently aligned for the hardware. Such a screen can come up skewed when promoted.

- Coercion of displays in back to match a promoted or explicit DblNTSC or DblPAL front display may result in a lower resolution mode for unsuitably aligned back displays.

"1x" modes require 16-pixel (word boundary) alignment of each scan-line. "2x" modes require 32-pixel (longword boundary) alignment, while "4x" modes require 64-pixel (double-longword boundary) alignment. Here is a short reference:

- 140 ns pixels (lores in 15 kHz modes, extra-lores in 31 kHz modes)
  1-8 planes require 1X

- 70 ns pixels (hires in 15 kHz modes, lores in 31 kHz modes)
  1-4 planes require 1X
  5-8 planes require 2X

- 35 ns pixels (super-hires in 15 kHz modes, hires in 31 kHz modes)
  1-2 planes require 1X
  2-4 planes require 2X
  5-8 planes require 4X

As the graphics.library AllocBitMap() function takes care of allocating suitably-aligned BitMaps for you, you do not need to worry about alignment when using modern system calls.

- The AA hardware does not allow dual-playfield non-interlaced screens to be scan-doubled, so they will appear half as tall as their non-promoted counterparts.
- Like earlier chipsets, the AA chipset still supports eight sprites. In much the same way as ECS and original chipsets lose sprites when overscan is increased, many of the new modes have insufficient spare cycles to fetch data for these sprites. A promoted screen may have fewer sprites left than the corresponding 15 kHz mode, meaning that some sprites other than the pointer sprite may vanish.

Note - If the system is aware that additional sprites are needed, it will attempt to drop a display's bandwidth (if possible) to allow additional sprites. To cause this, either GetSprite() your sprites before opening your display, or RemakeDisplay() after doing your GetSprite() calls.

- There is currently no 31 kHz mode having 1280 pixels per line. That would require 17.5 ns pixel speeds, which is twice what the AA chipset is capable of. Therefore, SuperHires screens are promoted to 640 pixel-per-line screens, which generally can scroll. This is required for systems that have a VGA-only monitor, but otherwise might not be the desired way to display this mode. To prevent promotion of

SuperHires to Hires, an application could explicitly ask for NTSC or PAL SuperHires. However, keep in mind that future chipsets may be capable of de-interlacing 1280-wide displays, and this would be defeated if NTSC or PAL is explicitly requested. Also remember that a user with a VGA-only monitor cannot display these modes. The V39BestModeIDA() function may be used to determine the best available mode.

- Applications that specifically request the NTSC or PAL monitor when opening a display will receive non-promoted PAL or NTSC, suitable for genlocking, etc. For interlace modes, this will not be a flicker-free display. Therefore, productivity and design applications that wish to offer an explicit PAL/NTSC choice should use the display database or 2.1 ASL screen mode requester to allow the user to choose a flicker-free display mode that is available on their system.

Other Compatibility Issues

- Library Vector Offsets and SetFunction(). In general, some new LVOs supersede old ones. While the old ones still work, software that calls SetFunction() based on the old LVOs may no longer catch what they were hoping to. An example from V37 was people who called SetFunction() on AutoRequest(). Most system requests go through EasyRequest() now. A V39 example would be SetWindowPointer() replacing SetPointer().
- Test for Memory by Poking Breaks. Apparently, some games test for RAM at $200000 by writing a pattern there and immediately reading it back. On the A1200 bus, such games will be fooled. Never test for RAM by poking/peeking. Always use Exec's memory allocation functions, or the Exec TypeOfMem() function which can tell the caller whether (and what type of) memory exists at any particular address.
- Changes to support for width-scaling of fonts Under V37, different widths of one YSize of a scalable font could be opened with a tagged OpenDiskFont() by first running a patch called setpatchwtam. Under V38, this patch program does nothing, and an incorrect already-opened width may be returned. Under V39, the correct requested width should again be returned, and these loaded width-scaled fonts are both hidden from AvailFonts() and should not be accidentally provided if application just does OpenFont() for the same YSize.
- Tablet driver writers should test that their tablets now work with console drag selection (known not to work in the 2.04 OS).
- The V39 Iffparse.library is not 1.3 compatible. Use the V37 library instead.

- Preferences file format changes The format of some Preferences files have changed of necessity to support new system capabilities, and these files may continue to change. The documentation which has been provided only shows some chunks which may be encountered in a system Preferences file. There is no guarantee that new chunks will not be added, or that the current chunks will continue to be used. Do not read or write system Preferences files. Instead use other provided system mechanisms for reading and setting such items (for example, use device-specific commands or structures for controlling device preferences, functions such as QueryOverscan() and LockPubScreen() for determining display characteristics, etc.)
- Self-modifying, copied, and directly loaded code Self-modifying code without proper cache control has been breaking since the 68020 was introduced. The larger caches of the 68030 and 68040 processors make it even more necessary to use the exec cache control functions such as CacheClearU() which have been available since V37. The cache control functions make it possible to flush the processor caches after modifying code in place, copying code to a different memory address, or placing code into memory via any mechanism that bypasses LoadSeg. Cache-control functions are also provided for DMA controllers which DMA data into memory.
- IFF code. Older IFF sample/example code contains many hardcoded limits and accesses some structures which may no longer be accessed directly. The older code (and even the code listed in the *Amiga ROM Kernel Reference Manual: Devices*, 3rd Edition) also would not load more than 32 color registers. The new IFF code, NewIFF39.lzh, attempts to use the latest system functions wherever possible in a conditional backward-compatible manner. The new code provides support for the full AA color palette, arbitrary display modes, the clipboard, overscan and autoscroll screens, and loading/saving of 24-bit ILBMs. See the ILBM Compatibility notes for additional information on ILBM compatibility and support of AA features.
- SetMap is gone. The SetMap command (that set the keymap for the whole system) has been replaced by SetKeyBoard, that only sets the keymap for the current console. However, if you absolutely require it, the V37 SetMap command still works.
- Speech is gone. The narrator.device and translator.library are not part of V38 and V39. Related handlers and utilities are also gone. Therefore, new machines shipped with V38 or V39 do not have speech.
- Pre/Post DMA Cache functions and 68040 For those of you with DMA devices that need to use the CachePreDMA() and CachePostDMA() calls: Be careful as incorrect use of these calls may look like they work

fine in 68000/68020/68030 systems but may cause strange system behavior and even major system slowdown on 68040 systems.

The correct way to use these calls is:

```
original_length=length;
CachePreDMA(address,&length,0);

/* Optional, multiple ones here for breaking up DMA operations within the single
 * larger DMA block defined in the first call to CachePreDMA()
 */

CachePreDMA(....,....,DMA_Continue);

/*
 * Very important to call CachePostDMA() with the exact same values as
CachePreDMA()
 */

CachePostDMA(address,&original_length,0)
```

Also, you must make sure they match up. If they do not, the system will not be able to figure out that the DMA that was starting has finished. Internally, the OS needs to track these things for the 68040 and higher processors and may need to track them even more in the future.

- An upcoming release of 3.0 will support the dos.library SetVBuf() function. From V36 to V39 this call did nothing. It will now properly change buffering modes, sizes, etc. For programs using Dos buffered I/O calls (FGetC, FPutC, FRead, etc.) this can make a major improvement in I/O performance if the buffer size is increased (and if the buffering mode is changed when writing to BUF_FULL).

For additional information on OS changes, take particular note of all BUGS and NOTES entries in the system autodocs.

Taking Advantage of New Features

By simply using system features and functions which were introduced in Release 2, you can write adaptable software that can automatically grow and support many new system software and hardware capabilities.

Add a bit of conditional code to treat color guns (R, G, and B) as at least 8-bits each, and to conditionally use the new 32-bit V39 color setting and getting functions, and your application can provide full AA palette support.

Here are some strategies for adapting to new system capabilities.

- Use the asl.library requesters if available. They are localized, they cut down on the amount of code you have to maintain, and they grow with the system. 2.0 Workbench licensees may get a free amendment to distribute the 2.1 asl.library which contains the screen mode requester.
- Adapt to previously impossible deeper modes. Properly written software can use the calling syntax of the current graphics and Intuition library functions to open the screen modes with larger numbers of bitplanes. These modes are simply deeper bitplane versions of existing modes: Lores 6,7, and 8 bits; Hires 5,6,7, and 8 bits; SuperHires 3,4,5,6,7, and 8 bits; and VGA 3,4,5,6,7, and 8 bits.

Note that the display database must be checked to see if the user's machine supports the desired mode and number of bitplanes.

Intuition's screen functions can be used to open these previously impossible screens, but programs cannot rely on Intuition to throw out all "bad" possibilities. For example, V36 Intuition will open an 8-bitplane Lores screen, but this is a waste of memory if the new chipset is not present since the 3 extra bitplanes will simply go unused. Programs will therefore need to explicitly check the display database to find out what the chips support. In V39, Intuition will check the chips for you and fail to open a screen which is "too deep." There is a new SA_ErrorCode value, OSERR_TOODEEP.

> *Note on Mode IDs.* The currently defined display database mode IDs have all been moved from <graphics/displayinfo.h> to the new include file <graphics/modeid.h>. Do not attempt to create your own display mode IDs by OR'ing together monitors and modes unless the include file explicitly allows it. Do not try to draw conclusions about display characteristics by interpreting bits in the display mode ID. Instead, pass the mode ID to the display database functions to discover the display characteristics (see the example below and other examples in Amiga Mail, DevCon Notes, *Amiga Rom Kernel Reference Manual: Libraries*, and the new screen mode requester example).

The complete example listed below will check the display database entry for a LORES screen and open a screen with the maximum allowable number of bitplanes (assuming there is enough Chip memory). On an Amiga with any chipset up to and including ECS, this code will open a 5-bitplane screen. On a machine with the AA chipset, the screen will be 8 bits deep. The code will then draw one vertical line for each available color using Intuition's preferred palette (which can be set by the user with Preferences). The example requires V36 or a later version of the OS.

```
/* maxdepthlores.c  */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/displayinfo.h>
#include <dos/dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/dos_protos.h>

#include <stdio.h>
#include <stdlib.h>

struct Library *IntuitionBase;
struct Library *GfxBase;

void Quit(char *whytext,LONG failcode)
{
if(*whytext)
   printf("%s",whytext);

if (IntuitionBase)
   CloseLibrary(IntuitionBase);

if (GfxBase)
   CloseLibrary(GfxBase);

exit(failcode);
}

void main(void)
{
ULONG modeID = LORES_KEY;
DisplayInfoHandle displayhandle;
struct DimensionInfo dimensioninfo;

UWORD maxdepth, maxcolors;
ULONG soerror = NULL,colornum;
struct Screen *screen;

if ((GfxBase = OpenLibrary("graphics.library",36))==NULL)
      Quit("graphics.library is too old <V36",RETURN_FAIL);

if ((IntuitionBase = OpenLibrary("intuition.library",36))==NULL)
      Quit("intuition.library is too old <V36",RETURN_FAIL);

if ((displayhandle=FindDisplayInfo(modeID))==NULL)
      Quit("modeID not found in display database",RETURN_FAIL);

if (GetDisplayInfoData(displayhandle,(UBYTE *)&dimensioninfo,
                       sizeof(struct DimensionInfo),DTAG_DIMS,NULL)==0)
      Quit("mode dimension info not available",RETURN_FAIL);

maxdepth=dimensioninfo.MaxDepth;
printf("dimensioninfo.MaxDepth=%d",(int) maxdepth);

if (screen=OpenScreenTags(NULL, SA_DisplayID ,modeID,
                          SA_Depth, (UBYTE) maxdepth,
                          SA_Title, "MaxDepth LORES",
                          SA_ErrorCode, &soerror,
                          SA_FullPalette, TRUE,
                          TAG_END))
```

```
    {
    /* Zowee! we actually got the screen open! now let's try drawing into it. */
    maxcolors=1<<maxdepth;
    printf("maxcolors=%d",(int) maxcolors);

    for(colornum=0;colornum<maxcolors;++colornum)
        {
        SetAPen(&(screen->RastPort),colornum);
        Move(&(screen->RastPort),colornum,screen->BarHeight + 2);
        Draw(&(screen->RastPort),colornum,screen->Height - 1);
        }

    Delay(TICKS_PER_SECOND * 6);
    CloseScreen(screen);
    }
else
    {
    /* Hmmm.  Couldn't open the screen.  maybe not enough CHIP RAM?
     * Maybe not enough chips! ;-)
     */
    switch(soerror)
        {
        case OSERR_NOCHIPS:
            Quit("Bummer! You need new chips dude!",RETURN_FAIL);
            break;
        case OSERR_UNKNOWNMODE:
            Quit("Bummer! Unknown screen mode.",RETURN_FAIL);
            break;
        case OSERR_NOCHIPMEM:
            Quit("Not enough CHIP memory.",RETURN_FAIL);
            break;
        case OSERR_NOMEM:
            Quit("Not enough FAST memory.",RETURN_FAIL);
            break;
        default:
            printf("soerror=%d",soerror);
            Quit("Screen opening error.",RETURN_FAIL);
            break;
        }
    Quit("Couldn't open screen.",RETURN_FAIL);
    }

Quit("",RETURN_OK);  /* clean up and exit */
}
```

- Open on Workbench. One of the easiest ways to help a given program peacefully coexist with new hardware is to allow it to open on the Workbench screen (or on any public screen). Software written this way should be robust enough that it can find out from the system any information it might need about the display environment, and, if it understands the kind of screen that it's on, use the appropriate graphics.library calls to manipulate its windows.

Here's some example code that determines the depth of the default public screen and opens a window on it. If it were part of a real application, this code would tell how many colors the screen has.

Add some conditional V39 palette sharing code and you can even have your own color registers to use, if available, or closest matching register to share.

```
/* depthawarevisitor.c */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/displayinfo.h>
#include <dos/dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/dos_protos.h>

#include <stdio.h>
#include <stdlib.h>

struct Library *IntuitionBase;
struct Library *GfxBase;
struct Screen  *screen = NULL;

void Quit(char *whytext, LONG failcode)
{
if(*whytext)
    printf("%s",whytext);

if (screen)
   UnlockPubScreen(NULL, screen);

if (IntuitionBase)
   CloseLibrary(IntuitionBase);

if (GfxBase)
   CloseLibrary(GfxBase);

exit(failcode);
}

void main(void)
{
struct Screen *screen;
struct DrawInfo *drawinfo;
struct Window *window;
UWORD depth;

if ((GfxBase = OpenLibrary("graphics.library",36))==NULL)
   Quit("graphics.library is too old <V36",RETURN_FAIL);

if ((IntuitionBase = OpenLibrary("intuition.library",36))==NULL)
   Quit("intuition.library is too old <V36",RETURN_FAIL);

if (!(screen = LockPubScreen(NULL)))
   Quit("Can't lock default public screen",RETURN_FAIL);

/* Here's where we'll ask Intuition about the screen. */
if((drawinfo=GetScreenDrawInfo(screen)) == NULL)
   Quit("Can't get DrawInfo",RETURN_FAIL);
depth=drawinfo->dri_Depth;

/* Because Intuition allocates the DrawInfo structure, we have to tell it when
 * we're done, to get the memory back.
 */
FreeScreenDrawInfo(screen, drawinfo);
```

```
/* This next line takes advantage of the stack-based amiga.lib version of        *
OpenWindowTags().
 */
if (window = OpenWindowTags(NULL, WA_PubScreen ,screen,
                            WA_Left, 0,
                            WA_Width, screen->Width,
                            WA_Top, screen->BarHeight,
                            WA_Height, screen->Height - screen->BarHeight,
                            WA_Flags, WINDOWDRAG|WINDOWDEPTH|WINDOWCLOSE|
                                      ACTIVATE|SIMPLE_REFRESH|NOCAREREFRESH,
                            WA_Title, "Big Visitor",
                            TAG_END))
    {

    printf("depth=%d",depth);

    /* All our window event handling might go here */

    Delay(TICKS_PER_SECOND * 10);

    /* Of course, some other program might come along and change the attributes of
     * the screen that we read from DrawInfo, but that's a mean thing to do to a
     * public screen, so let's hope it doesn't happen.
     */

    CloseWindow(window);
    }

Quit("",RETURN_OK); /* clean up (close/unlock) and exit */
}
```

- If necessary, use tag-extended older structures with 1.3 functions. If you are not yet able to drop support for 1.3 systems, use tag-extended structures such as ExtNewScreen and ExtNewWindow to provide V37 and V39/AA enhanced capabilities with 1.3-compatible code. See the Intuition chapters of the *Amiga ROM Kernel Reference Manual: Libraries* for examples that use the tag-extended structures. An example of such coding is the NewIFF39 code.

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors		Bandwidth (see note 1)
LORES	6	64		1x
(320 x 200 NTSC	7	128		1x
or 320 x 256 PAL)	8	256		1x
	8 HAM	256,000+	(see note 2)	1x
HIRES	5	32		2x
(640 x 200 NTSC	6 EHB	64	(see note 3)	2x
or 640 x 256 PAL)	6 HAM	4096	(see note 4)	2x
	6	64		2x
	7	128		2x
	8	256		2x
	8 HAM	256,000+	(see note 2)	2x
SUPERHIRES	1	2	(see note 5)	1x
(1280 x200 NTSC	2	4	(see note 5)	1x
or 1280 x 256 PAL)	3	8		2x
	4	16		2x
	5	32		4x
	6 EHB	64	(see note 3)	4x
	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x
MULTISCAN	1	2	(see note 5)	1x
(160, 320, 640 x 480	2	4	(see note 5)	1x
non-interlaced)	3	8		2x
	4	16		2x
	5	32		4x
	6 EHB	64	(see note 3)	4x
	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors		Bandwidth (see note 1)
SUPER72	1	2	(see note 5)	1x
(71 Hz refresh and	2	4	(see note 5)	1x
23.21 kHz scan rate;	3	8		2x
200, 400, 800 x 300	4	16		2x
non-interlaced or	5	32		4x
200, 400, 800 x 600	6 EHB	64	(see note 3)	4x
interlaced)	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x
EURO72	1	2	(see note 5)	1x
(69 Hz refresh and	2	4	(see note 5)	1x
29.32 kHz scan rate;	3	8		2x
160, 320, 640 x 480	4	16		2x
non-interlaced)	5	32		4x
	6 EHB	64	(see note 3)	4x
	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x
DOUBLENTSC	1	2	(see note 5)	1x
(58 Hz refresh and	2	4	(see note 5)	1x
27.66 kHz scan rate;	3	8		2x
160, 320, 640 x 280	4	16		2x
scan-doubled or	5	32		4x
160, 320, 640 x 480	6 EHB	64	(see note 3)	4x
non-interlaced or	6 HAM	4096	(see note 4)	4x
160, 320, 640 x 800	6	64		4x
interlaced)	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors		Bandwidth (see note 1)
DOUBLEPAL	1	2	(see note 5)	1x
(58Hz refresh and	2	4	(see note 5)	1x
27.66 kHz scan rate;	3	8		2x
160, 320, 640 x 256	4	16		2x
scan-doubled or	5	32		4x
160, 320, 640 x 512	6 EHB	64	(see note 3)	4x
non-interlaced or	6 HAM	4096	(see note 4)	4x
160, 320, 640 x 1024	6	64		4x
interlaced)	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x

Notes

1 - The bandwidth number describes the relative amount of fetch bandwidth required by a particular screen mode. For example, a 5-bit deep MultiScan screen requires 4x fetch bandwidth while a 1-bit MultiScan screen requires only 1x. This means the hardware has to move data 4 times faster on the deeper screen. To be able to move data at these higher rates, the higher bandwidth modes require data to be properly aligned in Chip memory that is fast enough to support the bandwidth. Specifically, bandwidth factors of 1x require data to be on 16-bit boundaries, factors of 2x require 32-bit boundaries, and factors of 4x require 64-bit boundaries.

Restrictions like these are the best reason to use the system allocation functions whenever data is being prepared for the custom hardware. It is not guaranteed that all machines that have the new chip set will also have memory fast enough for the 4x modes. Therefore, the only way to know whether or not the machine will support the mode you want is to check the display database.

2 - New 8-bit HAM mode uses the upper 6 bits as an offset into a table of 64 24-bit base color registers and the lower 2 bits for modify mode control. This mode could conceivably allow simultaneous display of more than 256,000 colors (up to 16.8 million, presuming a monitor and screen mode with enough pixels). Please note that while the register planes and control planes are internally reversed in 8-bit HAM (the modify control bits are the two LSBs instead of the two MSBs), programs using graphics library and Intuition will not have to deal with this reversal, as it will be handled automatically for them.

3 - This is like the original EHB mode, but in new resolutions. It uses 5 bits as an offset into a table of 32 color registers plus a sixth bit to yield an additional 32 colors that are 1/2 as bright.

4 - This is like the original 6-bit HAM mode, but in new resolutions. It uses the lower 4 bits as an offset into a table of 16 color registers and the upper 2 bits for modify mode control. This mode allows simultaneous display of 4096 colors.

5 - These modes are similar to the VGA and SuperHires modes of the ECS chipset except that they are not restricted to a nonstandard 64-color palette.

◆

Overview of V39 Graphics

by Spencer Shanson

The Advanced Amiga Chip set (AA) redefines graphics performance on the Amiga by adding many new display modes, new features and more color to the Amiga platform. This article presents an overview of how the new V39 graphics library has implemented theses new features in the system software.

In V39, the graphics library has changed in the following areas:

- Support for new AA display modes
- Double buffering
- Retargettable sprite and screen (ViewPort) functions
- Palette sharing
- Bitmap functions
- Interleaved BitMaps
- Display mode promotion, coercion and selection
- RTG compatible RastPort functions.

The original Amiga graphics library exposed many device-dependent details to the application programmer. Because of this, introduction of new graphics devices has been slowed, and application support of new features has been delayed. To reverse this trend, no features have been added to the new graphics system which cannot be kept for the future. Thus, when newer graphics systems are introduced, system software will need fewer changes, and applications will be ready to automatically use new capabilities. Thus, parameters such as number of bits per pixel, resolution, color palette size, etc., are either variable or have been made very large.

Compatibility

The AA chips are register level compatible with the old and ECS chips at boot up time. However, when new AA modes are enabled and displayed, and a game then takes over the screen display without informing the OS of it, some registers may be in incompatible settings.

Two approaches will reduce this:

1) Old Programs which boot with their own custom boot block (games) will have AA features disabled unless they are specifically asked for. This should ensure transparent compatibility for all bootable games.

2) It will be possible to disable AA features for non-compatible programs. This will be done via the "BootMenu" which is available at system boot time. New options in this menu will allow disabling of AA, disabling of ECS, and switching of PAL and NTSC.

For bootable games that want AA features, use the V39 graphics function SetChipRev(). This lets you upgrade the Amiga to be any chip revision you need, and handles the extra housekeeping that graphics needs to operate the selected chipset (such as updating the graphics database). The only restriction is that it is not possible to downgrade the setting from AA down to ECS or A.

Get/Set Functions

This release of graphics.library will attempt to ease the (future) transition to Retargetable Graphics. New functions are provided to do some operations in a more device-independent manner. This will help when we have to support foreign graphics devices, more than 8 bits per pixel, chunky pixels, and true-color displays.

These new "Get/Set" functions will allow device independent access to fields in the RastPort structure which were only previously manipulable by direct structure access.

`VOID SetAPen(rp, pen)`	Sets the current APen value
`ULONG GetAPen(rp)`	Return current APen value
`VOID SetBPen(rp, pen)`	Sets the current BPen value
`ULONG GetBPen(rp)`	Return current BPen value
`VOID SetDrMd(rp, mode)`	Sets the current DrawMode
`ULONG GetDrMd(rp)`	Return current DrawMode
`VOID SetOPen(rp,c)`	Was a macro. Now use the SetOutlinePen() function, or the SafeSetOutlinePen() macro.
`ULONG GetOPen(rp)`	Return current area outline pen
`VOID SetWrMsk(rp,m)`	Was a macro. Now use the SetWriteMask() function, or the SafeSetWriteMask() macro.
`ULONG GetWrMsk(rp)`	Description of this function is left as an exercise for the reader.

`VOID SetABPenDrMd(rp,apen,bpen,drmd)`
Sets the APen, BPen, and DrawMode for a RastPort. Faster than separate calls.

`VOID SetRPAttr(rp,taglist)`

You asked for it so here it is. Lets you set many of the RastPort attributes with one tag-based call. Here are the attributes currently supported:

RPTAG_Font	Font for Text() RPTAG_SoftStyle style for text (seegraphics/text.h)
RPTAG_APen	Primary rendering pen
RPTAG_BPen	Secondary rendering pen
RPTAG_DrMd	Drawing mode (see graphics/rastport.h)
RPTAG_OutLinePen	Area Outline pen
RPTAG_WriteMask	Bit Mask for writing.
RPTAG_MaxPen	Maximum pen to render (see SetMaxPen())

`VOID GetRPAttr(rp, taglist)`

No prizes for guessing what this does. It supports the extra tag RPTAG_DrawBounds. This tag is passed with a pointer to a rectangle structure. The returned rectangle structure will contain the bounds of the clipping regions inside the RastPort. This can be used to optimize drawing and refresh.

Color Map Functions

The color palette in AA is different in a lot of ways from the ECS one:

- It has 24 bits per entry, plus one bit to select transparency.
- There are 256 entries which is enough for many programs running on the same screen to share the palette.

All colors are now specified as 32 bit left-justified fractions, and are truncated based upon the number of bits that the hardware is capable of displaying.

There are now ...RGB32() functions to replace the ...RGB4() functions. These all work in 32-bits per gun, irrespective of the device the colors are intended for. Devices that cannot handle the color resolutions will truncate the colors to the most significant n bits. That is why it is important to duplicate the most significant n bits throughout the 32-bit resolution. For example, pure white should be treated as:

```
R = 0xffffffff, G = 0xffffffff, B = 0xffffffff
```

and not

```
R = 0xf0000000, G = 0xf0000000, B = 0xf0000000
```

The new color palette functions allow for multiple applications to coordinate their access to the palette. This allows applications to, for instance, dynamically remap pictures to match the color palette of the Workbench screen, display animations in windows, etc.

ObtainPen() will attempt to obtain a free palette entry for use by your program, and then set the pen number to the RGB value you specify. A pen can be either shared or exclusively for your own use. If shared, then you should not modify the RGB value of the pen once the pen is obtained, because other applications may use that same pen, and will be relying on its color.

ObtainBestPen() is a tag-based function that will attempt to find the pen in the ColorMap which is closest to the specified RGB value, and then return a shared pen to you. There are (currently) four levels of precision to which the RGB value must match. PRECISION_EXACT asks for an exact match of the RGB value. The other three are, in descending order of precision, PRECISION_IMAGE, PRECISION_ICON, and PRECISION_GUI. If there is no pen in the ColorMap with the required RGB value to the specified precision, and there are unallocated pens, then a new pen is reserved as shared, and its RGB value set to the value you requested.

Note, there is no way to physically stop you from changing the colors of shared pens, but it's just not done, so don't do it. Pens obtained either with ObtainPen() or ObtainBestPen() should be released back to the system with ReleasePen() when no longer used.

All the palette-sharing functions require a struct PalExtra to be attached to the ColorMap. This is done automatically for all Intuition screens, but if you are not using Intuition, then you will need to call AttachPalExtra() yourself. This allocates and attaches a PalExtra structure to the ColorMap. The PalExtra is deallocated by the FreeColorMap() function.

FindColor(cm,r,g,b,maxcolor) lets an application find the "closest" color to a given RGB value, independently of palette sharing. In fact, using SetRGB32CM and FindColor, you can perform color matching which is not associated with a display at all.

Bitmap Functions

These function exist because the new AA chips have alignment restrictions in high bandwidth modes. Changing InitBitMap() and AllocRaster() to obey these restrictions would have been very incompatible. Bitmaps created by AllocRaster with a multiple of 32 or 64 pixels per line will be compatible with high fetch modes (2x or 4x respectively). Incompatible ones will fall back to 1x mode, if 1x mode is capable of displaying the screen.

AllocBitMap() allocates an entire bitmap structure, and the display memory for it. AllocBitMap() allows you to use more than 8 planes, and also allows you to specify another bitmap pointer, thus telling the system to allocate the bitmap to be "like" another bitmap. A bitmap allocated in such a matter may be able to blit to this bitmap faster. Such a bitmap may be stored in a foreign device's local memory. Do not assume anything about the structure of a bitmap allocated in this manner. The size of a bitmap structure is subject to change in future graphics releases. Thus, you should use AllocBitMap()/FreeBitMap() for your raster allocation.

Sprite Functions

Graphics sprite functions (MoveSprite()) have been extended to understand large sprites, selectable sprite pixel resolution, and movement of scan-doubled sprites. Sprite positioning is no longer rounded down to lo-res pixel resolution. Applications will no longer have to "know" about the hardware-dependent format of sprite data.

The new sprite functions work with an ExtSprite structure, which is obtained with the tag-based AllocSpriteData() function. This function allows you to specify a BitMap for the image of the sprite, and tags to specify scaling factors to apply to the BitMap. The returned ExtSprite structure is then passed to GetExtSprite(), which is the V39 equivalent of the old GetSprite() function. There is no equivalent FreeExtSprite(0 function; FreeSprite() does the trick. The ExtSprite allocated with AllocSpriteData() is freed with FreeSpriteData().

The AA chip set imposes some limitations on sprites:

- All sprites in a ViewPort will be of the same resolution and width. There is no individual sprite resolution/width control. If the sprite you allocate is of a different width than Intuition's pointer, then Intuition is notified, and takes the appropriate steps to maintain the pointer imagery.

- In the programmable beam modes (namely Multiscan, Euro72, DblNTSC, DblPAL, and Super72), only sprite 0 is likely to be available; the other sprites are lost to bitplane DMA. MakeVPort() has code that detects which sprites have been reserved (with GetSprite() or GetExtSprite()), and if possible, will drop the display bandwidth. This will make available more DMA to the lower numbered sprites, at the expense of more bitplane DMA access. MakeVPort() will not drop the bandwidth if doing so would cause the loss of bitplanes in the display, so this is not always guaranteed to work for you.

All the AA sprite features are available through new tags for VideoControl():

VTAG_SPEVEN_BASE_SET/GET
: This sets the base color number of the even numbered sprites. The AA chip set allows odd numbered and even numbered sprites to use the colors of different 16-color banks, as opposed to the previous chip sets where all sprites used colours 16-31. Legal values to set the base are 0-255, but will be rounded down to the nearest multiple of 16.

VTAG_SPODD_BASE_SET/GET
: As VTAG_SPEVEN_BASE_SET/GET, only for the odd numbered sprites. For example, if the following tag list is passed to VideoControl():

```
struct TagItem[] = {
     {VTAG_SPEVEN_BASE_SET, 32},
     {VTAG_SPODD_BASE_SET, 144},
     {TAG_DONE}, };
```

then the sprites use the following colors:

Sprite	Colors
0	32 (transparent), 33,34,35
2	36 (transparent), 37,38,39
4	40 (transparent), 41,42,43
6	44 (transparent), 45,46,47
1	144 (transparent), 145,146,147
3	148 (transparent), 149,150,151
5	152 (transparent), 153,154,155
7	156 (transparent), 157,158,159

[Attached sprites use the palette settings of the odd numbered sprites.]

Normally, there are only (2 << depth) colours in a ColorMap (with the exception that Intuition screens always have a minimum of 32 colours). If you wish to set the sprite's colors to use pens that are outside of this range, then you should use the VTAG_FULLPALETTE_SET tag, which specifies that the ColorMap should contain entries for all possible colors (256). Therefore, the colors of these entries can be set with the usual ...RGB32() functions. Not setting VTAG_FULLPALETTE_SET in this case may cause unpredictable colors and enforcer hits.

VTAG_SPRITERESN_SET/GET

This allows you to set all the sprites in the ViewPort to one of 5 resolutions.

SPRITERESN_140NS:	All sprites have 140ns pixels.
SPRITERESN_70NS:	All sprites have 70ns pixels.
SPRITERESN_35NS:	All sprites have 35ns pixels.
SPRITERESN_DEFAULT:	All sprites have the Intuition default resolution.
SPRITERESN_ECS:	Compatible with ECS resolutions;140ns, except in 35ns display pixel modes (SuperHires), where the sprite pixels are 70ns.

VTAG_DEFSPRITERESN_SET/GET

For setting the default sprite resolution, as used by SPRITERESN_DEFAULT.

VTAG_BORDERSPRITE_SET/GET/CLR:

This sets the bordersprite option in the AA chipset for the ViewPort, which allows sprites to appear in the borders outside of the normal display window (DisplayClip), i.e., the area that is normally color 0. However, this does not apply to the horizontal blanking area.

VTAG_PF1_TO_SPRITEPRI_SET/GET:

All revisions of the Amiga chips have allowed the priorities of sprites to playfield to be set. Usually, the priority is such that the sprites always appear in front of the playfield. There is an entry in the ViewPort structure for altering this priority, but V39 provides this tag in preference to writing to the ViewPort structure.

VTAG_PF2_TO_SPRITEPRI_SET/GET:
As VTAG_PF1_TO_SPRITEPRI_SET/GET, only for playfield 2. On the Amiga, playfield 2 is the default playfield, and playfield 1 is only used in DualPlayfield modes!

Display Mode IDs and the Graphics Database

The graphics database has been extended where necessary for AA information, and these changes are limited to the DisplayInfo structure. The DisplayInfo->PaletteRange is only one WORD long, which is insufficient for the AA 24-bit colour resolution, and so has been superceded by three new entries called RedBits, BlueBits and GreenBits. These are one BYTE each, and show how many bits of precision is available for each colour gun. 255 bits each for red, green and blue should be enough for the foreseeable future (pun intended).

Some new DIPF_IS flags were added to the database, providing more information about each display mode ID.

DIPF_IS_SPRITES_ATT shows the mode supports attached sprites. The 35ns display modes on ECS could not support them.

DIPF_IS_SPRITES_CHNG_RES shows that the mode supports sprites that can change resolution, and so the VTAG_SPRITERESN_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SPRITES_BORDER shows that this mode supports border sprites, so the VTAG_BORDERSPRITE_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SCANDBL shows that this mode is scandoubled (each display line is repeated once), so that half as many lines take up the same physical area on the monitor. There should be no need for you to look at this bit.

DIPF_IS_SPRITES_CHNG_BASE shows that this mode supports sprite base colour offset changing, so that the VTAG_SPODD/SPEVEN_BASE_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SPRITES_CHNG_PRI shows that this mode supports changing the playfield to sprite priority, so that the VTAG_PF2/PF1_TO_SPRITEPRI_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_DBUFFER shows that this mode will work with the double-buffering ChangeVPBitMap() function. You should check this flag before using the double-buffering functions on the ViewPort.

DIPF_IS_PROGBEAM shows that this mode is a programmed beam-sync mode.

DIPF_IS_FOREIGN shows that this mode is not native to the Amiga chip set. Currently, no mode IDs will have this flag set, but under RTG many 3rd party display devices will. You may want to start checking for this flag now.

With the plethora of new display modes that are available under V38 and V39, it is now becoming harder and harder for both the application writer and the end user to know which display mode ID they need to use. This will be especially true under RTG when the application writer will have no way of knowing all the possible display mode IDs that are available on third party display devices. A function was added for V39 to alleviate this problem, and provide a means by which the system can calculate the best mode ID to use given a number of requirements, called BestModeID(). It takes the following tags:

BIDTAG_DIPFMustHave
The DIPF_ flags that this display mode ID must have set. Default is NULL, so there is no preference.

BIDTAG_DIPFMustNotHave
The DIPF_ flags that this display mode ID must not have set. For example, you may wish to ensure that only native Amiga modes are considered, so use BIDTAG_DIPFMustNotHave with DIPF_IS_FOREIGN. The default value is defined as SPECIAL_FLAGS, which is (DIPF_IS_DUALPF | DIPF_IS_PF2PRI DIPF_IS_HAM | DIPF_IS_EXTRAHALFBRITE) so you may need to OR your particular requirements with SPECIAL_FLAGS.

BIDTAG_ViewPort
An initializes ViewPort for which a mode ID is sought. For example, to find an interlaced version of a ViewPort:

```
ID = BestModeID( BIDTAG_ViewPort, ThisViewPort,
                 BIDTAG_MustHave, DIPF_IS_LACE,
                 TAG_END);
```

BIDTAG_NominalWidth
BIDTAG_NominalHeight
Together make up the aspect ratio of the desired mode ID. If specified, will override the Width and Height of the ViewPort passed in BIDTAG_ViewPort, or the NominalDimensionInfo of the ID passed in BIDTAG_SourceID. Defaults to 640x200.

BIDTAG_DesiredWidth
BIDTAG_DesiredHeight Nominal width and height of the returned mode ID.

BIDTAG_Depth
Mode ID must support at least this many bitplanes. Defaults to vp->RasInfo-> BitMap->Depth of the ViewPort passed in BIDTAG_ViewPort, or 1.

BIDTAG_MonitorID
The returned mode ID must belong to this monitor family.

BIDTAG_SourceID
BestModeID() will use characteristics of this mode ID, and override some of the characteristics with the other values in the taglist.

BIDTAG_RedBits
BIDTAG_BlueBits
BIDTAG_GreenBits
The mode ID must support at least these many bits for each color gun. Defaults to 4 bits each, so A2024 modes will not be considered.

As an example of its use, here is a portion of the code for the V39 CoerceMode() function (which is used by Intuition when coercing screens):

```
ULONG CoerceMode(struct ViewPort *vp, ULONG MonitorID, ULONG Flags)
{
...
    /* Coerce the ViewPort vp to the best fit ModeID in the monitor
     * MonitorID.
     */
    must = NULL;
    mustnot = SPECIAL_FLAGS;
    tag[t].ti_Tag = BIDTAG_ViewPort;
    tag[t++].ti_Data = vp;
    tag[t].ti_Tag = BIDTAG_MonitorID;
    tag[t++].ti_Data = MonitorID;

    if (GetDisplayInfoData(NULL, (APTR)&dinfo,
                           sizeof(struct DisplayInfo), DTAG_DISP, ID))
    {
        must = (dinfo.PropertyFlags & SPECIAL_FLAGS);
        mustnot = (SPECIAL_FLAGS & ~must);
        if ((Flags & AVOID_FLICKER) && (!(dinfo.PropertyFlags & DIPF_IS_LACE)))
        {
            /* we don't want lace if AVOID_FLICKER is set, and this
             * ViewPort is not naturally laced.
             */
            mustnot |= DIPF_IS_LACE;
        }
        tag[t].ti_Tag = BIDTAG_RedBits;
        tag[t++].ti_Data = dinfo.RedBits;
        tag[t].ti_Tag = BIDTAG_GreenBits;
        tag[t++].ti_Data = dinfo.GreenBits;
        tag[t].ti_Tag = BIDTAG_BlueBits;
        tag[t++].ti_Data = dinfo.BlueBits;
    }

    tag[t].ti_Tag = BIDTAG_DIPFMustNotHave;
    tag[t++].ti_Data = mustnot;
    tag[t].ti_Tag = BIDTAG_DIPFMustHave;
    tag[t++].ti_Data = must;
    tag[t].ti_Tag = TAG_DONE;

    return(BestModeIDA(tag));
}
```

As another example, consider an IFF display program with a HAM image, to be displayed in the same monitor type as the Workbench ViewPort.

```
ID = BestModeID(BIDTAG_NominalWidth, IFFImage->Width,
                BIDTAG_NominalHeight, IFFImage->Height,
                BIDTAG_Depth, IFFImage->Depth,
                BIDTAG_DIPFMustHave, DIPF_IS_AM,
                BIDTAG_MonitorID, (GetVPModeID(WbVP) & MONITOR_ID_MASK),
                TAG_END);
```

The definitions of the display mode ID keys have been moved from <graphics/displayinfo.h> to a new <graphics/modeid.h> file.

The include file <graphics/modeid.h> specifically says that the individual bits of the mode IDs should not be checked for any meaning, but that the database should be read to glean information about the mode ID. In order to maintain compatibility with old software, and make the application writer's job somewhat easier, I am willing to guarantee that any mode ID with the bit 0x00000800 set is a HAM mode, and any mode ID with the bit 0x00000080 set is an ExtraHalfBrite mode. These are the only bits that are guaranteed to mean anything in the mode ID itself. These bits correspond of course to the HIRES and EXTRA_HALFBRITE definitions in <graphics/view.h>.

Display Mode Promotion

Promotion is a software solution for the lack of hardware deinterlacing circuitry on the AA machines. With the promotion feature enabled in Icontrol, the default monitor (NTSC on NTSC machines, PAL on PAL machines), becomes DblNTSC on NTSC machine and DblPAL on PAL machines. There are a number of advantages and gotchas with this approach.

One advantage is that the graphics database is always truthful. Any enquiries about a default monitor mode ID will yield information relevant to whatever the default monitor happens to be at the time. This should not be a problem for any code written for V37 onwards, as the default monitor has always been either NTSC or PAL; now, there are just more possibilities. Another advantage is that V39-aware software that requires NTSC or PAL modes (e.g., video titling software), can now get such modes using specific NTSC or PAL mode IDs.

Here is a list of "gotchas" (that is, things to watch out for if you want display mode promotion to work correctly).

1) The default monitor is dynamic, and can change at any time. Therefore, do not cache any information about the default mode IDs, but read them from the database as you need them.

2) There is no equivalent in the Dbl... monitors to the SuperHires modes. The database LVOs sniff out SuperHires mode IDs for database enquiries, and map SuperHires mode IDs to the equivalent Dbl... Hires mode IDs if promotion is enabled.

3) Programs that rely on copper timings for UserCopperlists, such as SHAM displayers, may no longer work when promoted, because each line is shorter in time than the NTSC/PAL line. Therefore, there are less coppercycles per line.

4) Promoted ViewPorts have less sprites available than non-promoted ViewPorts.

5) For compatibility, we do not promote 1.3 style custom ViewPorts and Views (we check for the presence of a ViewExtra).

Miscellaneous

Interleaved screens have been added. These use a different layout of the graphics data in order to speed rendering operations and eliminate the annoying "color-flash" problem which is the hallmark of planar (as opposed to "chunky") display architectures. Set the BMF_INTERLEAVED flag when calling AllocBitMap().

Double buffering functions have been added. These allow applications to display very smooth, synchronized animation in fully an efficient "Intuition-friendly" manner. Call AllocDBufInfo() to allocate and initialize the DBufInfo structure, which is then passed to ChangeVPBitMap(). The double-buffering functions return up to two different types of messages. The first (dbi_SafeMessage) tells your program when it is safe to write to the old BitMap. The second (dbi_DispMessage) is sent when it is safe to call ChangeVPBitMap() again and be certain the new bitmap has been seen for at least 1 field. The autodocs for AllocDBufInfo() has example code showing how to safely double buffer with ChangeVPBitMap().

The DBufInfo structure should be freed with the FreeDBufInfo() function.

Due to the extra colours that need to be loaded in deeper AA screens, the gap between screens can now be greater than the three lines that Intuition traditionally kept. In fact, the number of lines between screens is variable, depending on the screen type and depth. A new function CalcIVG() (CalcInterViewPortGap) has been added to calculate the number of lines required by the copper to execute all the copper instructions before the display window is opened. Note however that CalcIVG() returns the true number of lines (rounded up to the next whole line) needed in ViewPort resolution units, but Intuition still maintains a gap of at least three lines between Screens. Therefore, when calling CalcIVG() to position screens in an Intuition environment, use the result of MAX((laced ? 6 : 3), CalcIVG(v, vp)).

There is one other caveat with respect to CalcIVG(). This function works by counting the number of copper instructions in the ViewPort->DspIns list, which is set up by MakeVPort(). If an Intuition screen is opened behind, then MakeVPort() is not called on that screen's ViewPort until it first becomes visible, so calling CalcIVG() with that screen's ViewPort will yield a result of 0.

Some operations have been sped up: RectFill() has been rewritten, WritePixel() uses the CPU (3x speedup) ScrollVPort is 10 times faster, and other optimizations.

~~Bugs~~ Anomolies

There is a big bug in the V37-V39 graphics hash-table code, which is used to associate a ViewExtra with a View; namely, only 8 Views can have ViewExtras attached to them. This has been fixed in the latest SetPatch and Kickstart.

ScrollVPort() and ChangeVPBitMap() have problems with 8-bit HAM mode. This has been SetPatch'ed.

Many other outstanding bugs have been fixed for V39.

Plans for 3.01

Some features that are planned for Release 3.01:

- ❑ An LVO for the games writers to handle color fades with user copperlists, and to allow independent scrolling of parts of the ViewPort for parallax scrolling games.
- ❑ Frame rate control for ChangeVPBitMap().
- ❑ Add tags to BestModeID() for overscan considerations.
- ❑ Still faster ScrollVPort (next beta).

Intuition 3.0

by Peter Cherna

Graphics and AA-Chipset Issues

Support for New Modes

Intuition now has direct support for the AA display modes, through extensions to old mechanisms and through new tags. This includes the ability to select higher resolutions and to set colors in better than 4 bits-per-gun. The graphics database and the ASL screen-mode requester are the definitive places to get information about what modes and depths are available or desired. You can specify higher depths using SA_Depth, and new display modes with SA_DisplayID. SA_Colors32 supplants SA_Colors for specifying colors with a higher precision than 4 bits-per-gun. For full details, see the section on new screen features.

Mode Promotion

The AA chipset provides some flicker-free display modes that are roughly equivalent to NTSC or PAL when output through a display-enhancer or flicker-fixing product. While the AA chipset provides these modes without the significant extra expense of a display-enhancer, some software help is required. Conversely, the display-enhancer lives on the video output, and is completely transparent to software.

To use promotion, the user needs to have a multiscan monitor. He needs to install the DblNTSC or DblPAL monitor into his devs:monitors drawer, and ensure that the "Mode Promotion" option in IControl Prefs is on (starting with 3.01, this setting will be on by default, eliminating this last step). When this is done, the graphics database entries for the "default" monitor will map to the most appropriate modes from the DblNTSC (or DblPAL) monitor, in place of the NTSC (or PAL) monitor.

Interlaced screens that are promoted are displayed using double-height non-interlaced modes. For example, 640x400 NTSC interlaced appears as 640x400 double-NTSC non-interlaced. Non-interlaced 15 kHz screens are promoted using a chipset feature called "scan-doubling," in which each scan-line is output twice, because the 200 or 256 lines of the original screen only fill half the screen when in 31 kHz modes.

Most applications which use the default monitor (either by explicitly using mode-IDs such as

HIRES_KEY or by using V34-style 16-bit mode descriptions) and which go through Intuition and have relatively ordinary display requirements will be successfully and transparently promoted, and their screen will be displayed flicker-free. Applications which refer to modes of explicit monitors (e.g., SA_DisplayID of NTSC_MONITOR_ID|HIRES_KEY) are never promoted. The preferred method for an application is to use the ASL screen-mode requester, which normally presents all explicit modes (including the modes of DblNTSC and DblPAL) and omits the modes of the "default" monitor.

Promotion and Compatibility

De-interlacing in hardware is expensive but transparent, since the Amiga chipset's operation is unchanged by the presence of such hardware. However, promotion can change the behavior of the system in a manner which may be incompatible with certain applications. These are:

- The overscan limits of DblNTSC (DblPAL) are a little less than the overscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to NTSC (PAL)).
- It may be harder to center a DblNTSC (DblPAL) screen on certain multiscan monitors than it was to center a de-interlaced NTSC (PAL) screen. (3.01 provides additional centering flexibility which basically solves this problem).
- An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists.
- The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignment currently required by AA). However, if the custom BitMap is an unusual width, it may not be sufficiently aligned for the hardware. Such a screen can come up skewed when promoted.

"1x" modes require 16-pixel (word boundary) alignment of each scan-line. "2x" modes require 32-pixel (longword boundary) alignment, while "4x" modes require 64-pixel (double-longword boundary) alignment. Here is a short reference:

- 140 ns pixels (lores in 15 kHz modes, extra-lores in 31 kHz modes)
  1-8 planes require 1X

- 70 ns pixels (hires in 15 kHz modes, lores in 31 kHz modes)
  1-4 planes require 1X
  5-8 planes require 2X

- 35 ns pixels (super-hires in 15 kHz modes, hires in 31 kHz modes)
  1-2 planes require 1X
  2-4 planes require 2X
  5-8 planes require 4X

As the graphics.library AllocBitMap() function takes care of allocating suitably-aligned BitMaps for you, you do not need to worry about alignment when using modern system calls.

- The AA hardware does not allow dual-playfield non-interlaced screens to be scan-doubled, so they will appear half as tall as their non-promoted counterparts.
- Like earlier chipsets, the AA chipset still supports eight sprites. In much the same way as ECS and original chipsets lose sprites when overscan is increased, many of the new modes have insufficient spare cycles to fetch data for these sprites. A promoted screen may have fewer sprites left than the corresponding 15 kHz mode, meaning that some sprites other than the pointer sprite may vanish.
- There is no 31 kHz mode having 1280 pixels per line. That would require 17.5 ns pixel speeds, which is twice what the AA chipset is capable of. Therefore, SuperHires screens are promoted to 640 pixel-per-line screens, which generally can scroll.
- Custom ViewPorts are not promoted, but a graphics-database aware application could open a ViewPort in DblNTSC or DblPAL.

Pointer Sprite Features

New Pointer Features

Intuition's handling of the pointer sprite has undergone significant rework for V39, partly to add support for AA sprites (35/70/140 ns sprite pixels, 16/32/64 pixels per sprite, scan-doubling of sprites), and partly to make some general improvement to Intuition.

The Intuition pointer now supports the various new sprite modes of the AA chipset. This includes 16, 32, and 64-bit wide sprites, as well as the sprite-pixel resolution control.

Intuition automatically positions the pointer sprite on screen pixel resolution boundaries, even if the pointer is in a lower resolution, with the exception that graphics.library only allows positioning the pointer on every second line of an interlaced screen.

For applications, there is a new boopsi class called "pointerclass," which is used to create Intuition pointer objects. The new <intuition/pointerclass.hli> include file contains definitions for the attributes that pointerclass supports, including:

- POINTERA_BitMap - BitMap to use for sprite imagery
- POINTERA_XOffset, POINTERA_YOffset - sprite hot-spot
- POINTERA_WordWidth - intended width in words of this pointer
- POINTERA_XResolution, POINTERA_YResolution - intended resolution of this pointer

The resolution can be any of the hardware resolutions (ECS-compatible, 140 ns, 70 ns, or 35 ns), but Intuition also adds software-managed choices for

- sprite resolution to match screen pixel resolution
- sprite resolution to be "always lores" (~320 pixels per line)
- sprite resolution to be "always hires" (~640 pixels per line)

See <intuition/pointerclass.hli> for full details.

There is a new pointer-control function called SetWindowPointerA(), which takes a window and a taglist. The tag values are as follows:

- WA_Pointer (APTR) - used to specify an application custom pointer, ti_Data points to an instance of "pointerclass" you typically obtain using NewObject(). If NULL, you are requesting the Preferences default pointer.
- WA_BusyPointer (BOOL) - if ti_Data is TRUE, this tag requests the Preferences default busy pointer.
- WA_PointerDelay (BOOL) - if ti_Data is TRUE, this tag requests that the change of pointer imagery be deferred for a short duration. This is very useful for an application which is about to be busy for an unknown but possibly very short duration. Such an application should request both the Preferences default busy pointer and the pointer-delay feature. If the application clears the pointer or sets another pointer before the delay expires, the pending pointer change is cancelled. This reduces

short flashes of the busy pointer caused by the application having brief intervals of busy-ness.

The same three tags are now recognized by OpenWindowTagList(), so you can now arrange for a window to open with a custom pointer (or standard busy pointer) already in place, and never see a brief flash of the default pointer.

The user can can now specify 32 pixel wide pointers in hires or lores using Pointer Preferences. Pointer Preferences supports a user-defined default pointer image as well as a user-defined busy pointer image.

Intuition blanks the pointer around size or imagery changes, to reduce ugly flashing that might otherwise result.

On an upbeat note, the notorious off-by-one error in sprite hot-spot position was eliminated from all new Intuition and Graphics calls. When using the new mechanisms, always specify the correct hot-spot offset. (The truth is, we tried to leave the error in, but couldn't agree whether the error should be one lores pixel or one sprite-resolution pixel.)

Sprite Compatibility

The old SetPointer() and ClearPointer() functions still work, giving compatible Intuition pointers. Likewise, the pointer imagery in devs:system-configuration will be used until a pointer.prefs file is received. However, due to growing complexity in the pointer subsystem, calling SetPointer() or ClearPointer() from within an input handler or inside Begin/EndRefresh() runs a risk of deadlocking. This has been kludged around, however:

> ***This is why we warn people to stick to simple rendering functions only!*** While inside LockLayerInfo(), LockLayer(), BeginRefresh(), BeginUpdate(), etc., and to not call Intuition or other high-level system functions inside of LockIBase(). As well, great care should be taken inside an input handler (it's generally best for the handler to signal a high-priority task which actually does the work). Don't be the application that dies under the next release when an inappropriate function that happened to work now deadlocks because its handling has become more sophisticated.

See the Autodocs for these functions for more information on what other system calls are OK to use with these functions.

Intuition and Graphics are involved in a scheme to maximize compatibility with older sprite-using applications. The AA chipset supports variable sprite resolution and width, but the setting affects all sprites. If an application requests a specific sprite width (through the old graphics.library/GetSprite() call or the new graphics.library calls (GetExtSpriteA() and ChangeExtSpriteA()), and Intuition's sprite is not compatible with that request, then graphics.library will blank Intuition's sprite and notify Intuition. Intuition will rebound by generating the most suitable pointer sprite which is compatible.

Using an attached sprite for the Intuition pointer was quasi-supported under 2.0. It no longer works.

The pointer information returned by GetPrefs() is no longer kept up-to-date, since the pointer data can exceed the storage space available in struct Preferences. (The ROM default pointer will be returned in all cases). (Like V37, V39 ignores the pointer data in calls to SetPointer() after the first one, for reasons such as this).

Pen-Sharing Support

Under V39, graphics.library has functions that let multiple applications share the pens in a palette (ObtainBestPen(), etc.). Palette sharing allows an application to gain exclusive access to some palette entries, which that application may then use or change as it sees fit. As well, an application may access a pen as shareable, meaning that other applications in need of a similar color may also be granted that pen value. Because these pens are shared among several clients, applications may not alter their color.

Intuition now uses and supports this pen-sharing scheme. For all types of screen, the sprite pens and pens found in the DrawInfo->dri_Pens are obtained as shareable. By default, all other pens are allocated as exclusive on behalf of the screen opener (this provides compatibility when visitor windows aware of the pen-sharing functions open on unaware public screens). Exclusive pens are for the screen owner's use only, and may be changed at the owner's will.

Screens that are aware of pen-sharing issues should set the {SA_SharePens,TRUE} tag, which instructs Intuition to leave all other pens unallocated. The Workbench screen is so marked, but pens 0 to 3 and ~0 to ~3 are also made shareable, for compatibility. Screens with SA_SharePens set to TRUE will have the new PENSHARED bit set in the screen->Flags field.

The application may then allocate pens as needed. A paint package, for example, would allocate all colors it uses as exclusive. Other applications might allocate several colors as

shared or exclusive. Since Intuition opens all public screens in private state, the application has a chance to allocate its colors before making the screen available to visitors (see SetPubScreenModes()).

Preferences now listens to only 8 colors for the BitMap, which Intuition will set as the first four and the last four colors of the Workbench or any "Workbench-like" screen (those having the SA_FullPalette or SA_LikeWorkbench attributes). When the SA_Pens pen-array is evaluated, pens are masked to the number of available colors. As well, special definitions of pen-number (PEN_C3, PEN_C2, PEN_C1, and PEN_C0) mean the complementary pen numbers of pens 0 to 3, regardless of depth.

The way the DrawInfo pens are determined is Intuition picks a default pen-array. Then, any pens you supply with SA_Pens override the defaults, up until the ~0 in your array. If the screen is monochrome or old-look, the default will be the standard two-color pens. If the screen is two or more planes deep, the default will be the standard four-color pens. If any explicit pens are specified, the default colors for new-look menus and the titlebar match the V37 colors. If the SA_Pens tag points at ~0, the new-look menu colors will be used.

If the screen has the SA_LikeWorkbench property, the default will be the user's preferred pen-array, now changeable through Preferences. There is a preferred pen-array for four colors, and one for eight or more colors.

Miscellaneous Graphics-Level Changes

Intuition places a blanking gap between sliding screens. The time spent in this gap is used to load the hardware color registers, display-mode registers, and bitplane-pointers in a clean way. Under ECS and prior, three non-interlaced lines were sufficient, and this amount was hard-coded. Under AA, this amount may increase, so logic was added to graphics.library (CalcIVG()) to determine this amount. Intuition now bases its inter-screen gap on the result of this call.

Under 2.0, when a screen was coerced, Intuition determined what display mode to use based on tables built into Intuition. This was too limiting, so graphics.library now has a CoerceMode() function to fulfill this responsibility. Intuition now uses this function. (Interested persons should see graphics.library/BestModeIDA(), which has wider application than CoerceMode()).

When the user accesses the screen's menus, buffer-swapping will stop. The ChangeScreenBuffer() call will return failure during this time. When the user finishes his menu selection, buffer-swapping will be possible again. Only a small subset of gadgets are supportable in double-buffered screens. These gadgets are those whose imagery returns to the initial state when you release them (e.g., action buttons or the screen's depth gadget). To use other kinds of gadgets (such as sliders or string gadgets) you need to put them on a separate screen, which can be an attached screen.

Windows with borders are not supportable on double-buffered screens. Double-buffered screens are expected to consist nearly entirely of custom rendering.

An example program illustrating double-buffering under Intuition, with menu-lending and an attached screen to hold two slider gadgets is provided.

Miscellaneous Screen Features

The new {SA_Interleaved,TRUE} tag allows applications to request that their custom or public screen have an "interleaved" BitMap. Interleaved BitMaps are built out of a single allocation, instead of one per bitplane. Further, the data is laid out as follows:

```
bitplane 0, scan-line 0
bitplane 1, scan-line 0
...
bitplane n, scan-line 0
bitplane 0, scan-line 1
...
```

(Contrast this to regular BitMaps, where each bitplane is contiguous.) The primary advantage of interleaved BitMaps is that blitting between them is cleaner, because color artifacting is eliminated. As well, multiple small blits are replaced by fewer large blits, saving on blitter setup time and improving blitter/processor overlap.

The primary disadvantage of interleaved BitMaps is that BitMap->BytesPerRow no longer means "how many bytes are in one row of one bitplane." This field continues to mean "how many bytes must be added to the current address to arrive at the same pixel one row down." See the notes for the compatibility talk for further details. (Screen grabbers and screen printers seem to be the primary victims of this change.)

For compatibility, the Workbench screen is non-interleaved if it opens before IPrefs has run. If opened or reset after IPrefs has run, the Workbench screen will be interleaved.

The new SA_LikeWorkbench tag gives you a screen having the same attributes as the Workbench screen, including depth, colors, pen-array, screen mode, etc. Individual attributes can be overridden through the use of tags. (SA_LikeWorkbench itself overrides things specified in the NewScreen structure). Attention should be paid to hidden assumptions when doing this. For example, setting the depth to two makes assumptions about the pen values in the DrawInfo pens. Note that this tag requests that Intuition *attempt* to open the screen to match the Workbench. There are fallbacks in case that fails, so it is not correct to make enquiries about the Workbench screen then make strong assumptions about what you're going to get.

OpenScreen() now allocates an appropriate-sized ColorMap for new modes. The SA_ColorMapEntries tag can override this, to let the application increase the ColorMap size if needed.

The SA_Draggable and SA_Exclusive tags are designed to help implement "game screens" that coexist with Intuition. {SA_Draggable,FALSE} allows the caller to make a screen non-draggable. {SA_Exclusive,TRUE} allows the caller to make a screen exclusive, meaning it will never share the display with another screen. Dragging down a screen that's in front of an exclusive screen won't reveal the exclusive screen. Although exclusive screens can autoscroll, but they can't be pulled down below their natural top (since nothing should be visible behind). Starting with 3.01, you can attach one or more exclusive screens with the SA_Parent and SA_xxxChild tags already described. Such screens form an exclusive family, and only coexist on the display with each other, not with other screens (exclusive or not).

Intuition and graphics.library now support over-sized scrollable A2024/Moniterm screens. While such screens can autoscroll and be dragged around, they can't be pulled down, since other screens would not be visible behind them (similar to the SA_Exclusive property, but dictated by the graphics database MonitorInfo.Compatibility field).

The new ScreenPosition() function extends the functionality of MoveScreen() to also include absolute screen positioning and optional movement of {SA_Draggable,FALSE} screens. The hope is that only the screen's opener (and not any commodity-type programs) would forcibly move a non-draggable screen. As well, ScreenPosition() allows you to specify a rectangle in screen coordinates which you wish to be made visible. Over-sized screens will be scrolled such that the rectangle you supply will be on the visible part of the display.

The new ScreenDepth() function unifies ScreenToFront() and ScreenToBack() while adding a flag allowing optional in-family depth-arrangement of attached screens. ScreenToFront(), ScreenToBack(), Amiga-M/N, screen depth-gadget action, and EasyRequest screen-popping all go through the ScreenDepth() LVO. Due to a bug, WBenchToFront() and WBenchToBack() do not yet do so, but this is expected to be fixed for 3.01.

OpenScreen() and the routines which open the Workbench now call OpenScreenTagList() through the LVO, which will allow some useful SetFunction()ing.

OpenScreen() now supports the new SA_BackFill tag, to install a LayerInfo backfill hook for the screen.

Intuition now ensures that the Workbench screen is at least 640x200. This is needed in order to safely allow lores (and other odd resolution) Workbench screens.

The original Screen structure has an embedded instance of a BitMap structure, which is unfortunate. When Intuition allocates a screen's BitMap (i.e., a non-CUSTOMBITMAP screen), it now uses AllocBitMap(), and copies the struct BitMap into &screen->BitMap. This is the direction to head for RTG, and it is needed now for double-buffering. Intuition internally now references the real BitMap (obtainable as screen->RastPort.BitMap) instead of &screen->BitMap. It is recommended that applications do the same.

The new SA_Colors32 tag can be used to provide 32-bit color information at OpenScreen() time. ti_Data points to a longword-array that Intuition will pass to LoadRGB32(). See the autodoc for LoadRGB32() for details on how to format the data.

The new SA_VideoControl tag allows an application to provide a taglist which Intuition will pass to VideoControl() after opening the screen. This can be useful to turn on border-sprites, for example.

The requested screen depth is now validated. Making a request for a screen too deep causes failure with a secondary error of OSERR_TOODEEP.

Enhancements to Windows

The window depth-gadget now determines whether a click should send the window to front or to back based on whether the window is obscured or not, rather than on whether it's the top layer.

The new ScrollWindowRaster() function implements ScrollRasterBF() at the Intuition level. You will receive an IDCMP_REFRESHWINDOW event if there is damage to your window. Damage will appear if obscured parts of a simple refresh window are scrolled into view. *Note: this area is not cleared!*

When you supply the "alternate-size" rectangle for zooming using the WA_Zoom tag, you can now specify (-1,-1) for the upper-left corner. This instructs Intuition to perform size-only

zooming. Wherever the window is placed, zooming will toggle size but not affect position (unless moving the window would be required to keep it on-screen). Using (-1,-1) under V37 is safe, and equivalent to using (0,0).

The Window->WindowPort is now allocated at OpenWindow() time, even if IDCMPFlags of zero are requested. This simplifies ModifyIDCMP() failure handling, because failure can no longer happen when you set the window->UserPort to your shared port, then call ModifyIDCMP() to turn on messaging. ModifyIDCMP() can now only fail if you ask it to create the UserPort.

The new OpenWindow() tag WA_NotifyDepth allows a window to request IDCMP_CHANGEWINDOW messages when that window is depth-arranged. These messages arrive with an IntuiMessage->Code value of `CWCODE_DEPTH` to distinguish them from V37-style IDCMP_CHANGEWINDOW messages (sent in response to window movement or resizing), which have a Code value of `CW_MOVESIZE`.

When inactive, window borders are filled with BACKGROUNDPEN instead of pen zero. There are presumably a few more places when pen zero is being used incorrectly, but they are hard to track down (ever try to search 1.5 megabytes of source code for all references to "0"?)

The WA_HelpGroup and WA_HelpGroupWindow tags allow the programmer to identify multiple windows of the same application, for purposes of gadget help processing. See the section on gadget help for details.

Enhancements to Gadgets and Imagery

Extended Gadget Structure

V39 introduces the ExtGadget structure as a compatible substitute that can be used wherever old Gadget structures can be found. An arbitrary gadget can be identified as an ExtGadget if the GFLG_EXTENDED bit in its Flags field is set. Never attempt to read any of the extended fields of a gadget if this flag is not set. Starting with V39, all instances of gadgetclass or its subclasses are ExtGadgets. The extended fields include a new longword worth of flags, and a bounding box.

Gadget Bounding Box and GM_LAYOUT

Until now, the only area that was defined for gadgets was the select box area. However, the imagery of a gadget often extends outside its select box. For example, the border of a string

gadget is often outside, as is its label. ExtGadgets have four new fields that describe the "bounding box." The bounding box can be used to allow relative size or position gadgets to work even if they have imagery outside the select box. The BoundsLeftEdge, BoundsTopEdge, BoundsWidth, and BoundsHeight fields of an ExtGadget are assumed valid if the GMORE_BOUNDS flag in the ExtGadget->MoreFlags field is set. Where Intuition wants to use a bounding box, but the gadget is not extended or does not have GMORE_BOUNDS set, the gadget select box will be used instead, which matches the V37 behavior.

The routine that manages erasing and redrawing GRELxxx gadgets during window resizing now bases its work on the gadget bounding box (if one is specified), instead of the gadget select box. This means that you can finally have GREL gadgets which have imagery (e.g., a gadget label) extending outside of the select box, and Intuition will correctly move or resize such a gadget.

As well, there is a new boopsi method for gadgets called GM_LAYOUT. If your gadget has any of the GREL properties, it will receive a GM_LAYOUT message when the gadget is first added (or the window containing the gadget is first opened), as well as whenever the window size changes. At GM_LAYOUT time, the gadget can change its gadget select box and/or bounding box. It can re-allocate or change its image dimensions if it likes. If it is a group-gadget, it can move its members around. To round this all out, there is a new "special relativity" property, GFLG_RELSPECIAL or GA_RelSpecial. Unlike the older GREL properties, this property doesn't affect the interpretation of the gadget box. It does allow your gadget to receive GM_LAYOUT messages, hence have arbitrary layout power.

The DevCon disks include a sample program called relspecial.c that implements a boopsi gadget whose size is kept at half the current size of the window, and which is centered in that window. The example makes use of GFLG_RELSPECIAL and GM_LAYOUT.

Gadget Help

Intuition now supports "gadget help". If a window enables this feature and the user passes the mouse over the bounding box of a gadget which has the GMORE_GADGETHELP property, then an IDCMP_GADGETHELP event will be sent. There is a corresponding boopsi GM_HELPTEST method which boopsi gadgets can use to refine their help-sensitivity areas or to delegate help-testing to member gadgets. Boopsi gadgets can also return values for the IntuiMessage Code field of the IDCMP_GADGETHELP message.

The gadget help feature may be turned on or off through the HelpControl() function.

The gadget help checking is optimized for performance. If the mouse is moving quickly, Intuition skips the check for gadget help. If Intuition discovers that the mouse is still over same gadget as the last one that sent gadget help, no new IntuiMessage is sent unless the gadget wants to report a different IntuiMessage->Code value.

When the mouse is over a GMORE_GADGETHELP gadget, the IDCMP_GADGETHELP message has an IntuiMessage->IAddress which points to the gadget. When the mouse is over the window but not over any help-aware gadget, the IAddress points to the window itself. When the mouse is not over the window, the IntuiMessage IAddress will be zero. Intuition will look "through" gadgets that do not have the GMORE_GADGETHELP property to see if some other gadget lies underneath.

Ordinarily, gadget help only applies to the active window. However, a multi-window application can mark all its windows as being in a group (using the WA_HelpGroup or WA_HelpGroupWindow tags), which makes Intuition test gadget help in all windows of the group when any one of them is the active one. There is a new utility.library function called GetUniqueID() which must be used to provide an ID for WA_HelpGroup. If you have only one window, there is no need to pass WA_HelpGroup. HelpControl() sets the state of gadget help for all windows of a group, and Intuition ensures that all windows of the same group are consistently set.

```
helpgroup = GetUniqueID();

for ( each window )
    {
    win[x] = OpenWindowTags(...,
                            WA_HelpGroup, helpgroup, TAG_DONE);
    }
```

Inactive windows whose WA_HelpGroup matches the active window's are also subject to gadget help testing. IDCMP_GADGETHELP messages are sent to the window the mouse is over. The IDCMP_GADGETHELP message with an IAddress of zero means the mouse is not over the active window or any other window of the same group. This particular message is always sent to the active window (which is not necessarily the window in your group that last got a message).

All system gadgets (e.g., close, drag, size, depth) have GMORE_GADGETHELP set, so GadgetHelp-aware applications can (almost must) provide help on them too. You can check the gadget->GadgetType & 0xF0 field for GTYP_CLOSE, etc. Later include files define GTYP_SYSTYPEMASK to be this value (0xF0).

The gadgethelp.c example on the DevCon disks illustrates the correct handling of IDCMP_GADGETHELP messages.

Gadget Support for ScrollRaster() Damage Handling

Intuition now notices and repairs damage when boopsi gadgets use ScrollRaster(). (Such damage occurs when the gadget is in a simple-refresh window and part of the scrolled area is obscured). Such gadgets must set GMORE_SCROLLRASTER in order to benefit from this magic repair feature. Note that ScrollWindowRaster() is for applications. Boopsi gadgets must not use ScrollWindowRaster(), but rather use ScrollRaster() or ScrollRasterBF().

Miscellaneous Gadget Features

There is an important new boopsi function in Intuition called DoGadgetMethodA(), which invokes the specified method, but includes a valid GadgetInfo structure if possible. SetGadgetAttrsA() has been re-implemented to go through DoGadgetMethodA(). Two SetGadgetAttrsA() bugs were fixed in the process. First, if a requester is off-window, it has no layer. SetGadgetAttrsA() of a gadget in such a requester wasn't being sent to the gadget, but now OM_SET is now sent with a GadgetInfo of NULL. Second, there was no locking around the call. DoGadgetMethodA() is preferable to a direct boopsi invocation (namely DoMethod()) because it offers both a valid GadgetInfo structure and arbitration around other gadget activity.

The default string edit hook now ignores Return or Enter keystrokes that have the repeat-qualifier set.

PROPNEWLOOK proportional gadgets in the borders of an active window have their knobs rendered in FILLPEN instead of SHINEPEN. When you click on the knob, they become SHINEPEN as before. This looks a lot better, is more consistent with inactive windows, and finally restores that all-important feedback to these prop gadgets.

Imagery Features

Under V37 and prior, Intuition handled ghosting a gadget by blasting a pattern of dots over its select area. This did not allow boopsi images to manage their own disabled rendering. For V39, Intuition has defined a new read-only attribute for image classes, called IA_SupportsDisable. If an image class returns `TRUE` in response to an OM_GET request of this attribute, it is asking to take responsibility for performing ghosting based on image state. When a gadget is first added to a window, Intuition will check its image to determine if it has the IA_SupportsDisable attribute. When such a gadget is disabled, Intuition skips its own disabled rendering, and draws the image using DrawImageState() using the `IDS_DISABLED` or the new `IDS_SELECTEDDISABLED` state.

The ROM boopsi image class for rendering frames, "frameiclass", now supports the standard frame types used by GadTools and recommended by the *Amiga User Interface Style Guide*, including the standard GadTools-style bevelled box, the GadTools string-gadget ridge, and the AppWindow icon drop-box specified by the Style Guide.

The imagery for the standard system arrows has been improved.

The GadTools checkbox and GadTools radio-button images are now marked as scalable, to allow GadTools to support scaled checkboxes and radio-buttons.

The basic gadget types (classic prop, bool, string) now support GFLG_LABELIMAGE.

Enhancements to Menus

New-Look Title Bar and Menus

On aware screens, the title-bar has a nicer appearance.

For V39, Intuition defines three additional pens in the DrawInfo, which are used to control the rendering of the screen title bar and menus. These pens are:

- BARDETAILPEN - pen used for details like text in the title bar and text or graphics in menus.
- BARBLOCKPEN - pen used to fill the solid areas of the title bar and menus.
- BARTRIMPEN - pen to use for the trim-line under the screen title bar.

It is intended that BARDETAILPEN and BARTRIMPEN should be black or dark, and that BARBLOCKPEN should be white or light-colored.

The handling of defaults is a bit involved because of compatibility issues. Applications that specify no SA_Pens array or ones who specify an SA_Pens array with at least one explicit pen provided get compatible defaults. Applications whose SA_Pens array consists solely of {~0} will get the new colors. Finally, screens which specify SA_LikeWorkbench will get the user's preferred pen-array, which allows the user to control the menu and title bar colors.

Under V37 and earlier, the menus themselves are drawn using the window's DetailPen and BlockPen, while the colors of the MenuItems are determined by the imagery (IntuiTexts or Images) chosen. For compatibility, old applications will have their menus rendered in V37-compatible colors. Applications will want to take advantage of new-look menus when

present, but they must request them through the {WA_NewLookMenus,TRUE} tag. This instructs Intuition to use BARDETAILPEN and BARBLOCKPEN for rendering the elements of your menus, instead of relying on the window's DetailPen and BlockPen. Note that the application (or any menu-building library) should ensure that the colors of struct IntuiText or struct Images used by MenuItems use matching pens. You can instruct GadTools to use the new colors by passing {GTMN_NewLookMenus,TRUE} to LayoutMenusA().

Intuition also ensures that the Amiga-key and checkmark symbols are colored to match the menu and are scaled to match the screen's font. If you are using a font other than the screen's font for the menus, you must create custom Amiga-key and checkmark symbols using "sysiclass". This image class recognizes a new tag, SYSIA_ReferenceFont, which you can use to set the size of a checkmark or Amiga-key symbol appropriately for your chosen font. You can then use the (V36) WA_CheckMark tag or the new WA_AmigaKey tag to override the imagery Intuition will use in the menus.

Per screen, the default Amiga-key and checkmark images used will be appropriately colored and scaled to the screen's font. (You can find pointers to their imagery in the DrawInfo structure for that screen).

HIGHCOMP menu items in new-look menus complement in such a way that pixels colored in BARDETAILPEN highlight into BARBLOCKPEN, and vice-versa.

The flag which indicates whether a window is using new-look menus is publicly readable (WFLG_NEWLOOKMENUS).

Other Enhancements

Drawing Tablet Support

As an input device, a drawing tablet poses special problems for Intuition and for the tablet driver writer. Some of the problems include getting auxiliary information such as pressure through the IntuiMessage channel and providing suitable scaling. As well, the absolute nature of tablet devices poses some interesting problems for features like autoscroll. V37 added a simple tablet input event, but this was not enough. There now is a new subclass of IECLASS_NEWPOINTERPOS called IESUBCLASS_NEWTABLET. This subclass solves tablet handling quite nicely. The tablet driver fills out a few tablet-oriented properties (like the current value and range in X and Y), and then submits the InputEvent to input.device. Later, Intuition establishes the active screen and the rectangle to which the tablet should scale itself, then calls back the driver through a hook. This allows the tablet driver to handle screen

resolution changes, oversized scrolling screens, pulled down screens, and attached screens. The tablet driver can scale according to some tablet preferences settings it manages (for example, preserve aspect, center, best fit horizontal, best fit vertical). Intuition supplies reasonable default scaling for simple tablet drivers that leave the hook NULL. See struct IENewTablet in <devices/inputevent.h>.

Windows that request the new WA_TabletMessages property receive extended IntuiMessages, which include a pointer to a TabletData structure. If this IntuiMessage originated from a tablet event, the TabletData pointer will be non-NULL, and the structure will have some information such as sub-pixel position. In addition, there is a pointer to a tag-list, and there are definitions for standard tags such as pressure, tilt, additional buttons, and a Z-coordinate. The definition and important comments on the TabletData structure can be found in <intuition/intuition.h>.

As an example, pressure is passed in with the TABLETA_Pressure tag. the pressure reading of the stylus. The ti_Data member is the pressure, which should be normalized to fill a signed long integer. Typical devices would not generate negative pressure, but the possibility is not precluded. The Preferences program shipped with a tablet driver for a pressure-sensitive device might offer two pressure sensitivity settings. The "contact threshold" would be the pressure below which no contact should be reported by the driver. This is the "zero point" for reported pressure. The "click threshold" would the pressure at which a button transition should be reported (by setting the InputEvent ie_Code to indicate a downpress of the select button). The tablet would send position/pressure events even when the pressure was below the click threshold (but above the contact threshold, of course).

When a tablet driver sends a new tablet event and the active window is tablet-aware, IDCMP_MOUSEMOVE events are sent to that window even if the pixel-level mouse-position is unchanged. This is to allow applications to hear changes in sub-pixel position or other parameters such as pressure. This means that tablet drivers must be careful to only send events when something actually changed.

If an input event is a tablet event, boopsi gadgets can get a pointer to the TabletData structure in the gpInput structure they receive through the GM_GOACTIVE and GM_HANDLEINPUT messages.

Miscellaneous Features

The new TimedDisplayAlert() function allows for alerts that time-out without user-intervention. Exec uses this new function to aid unattended operation of the Amiga, particularly in kiosk and video applications.

In struct Preferences, the unused WorkName[] field is now split into PrtDevName[], DefaultSerUnit, and DefaultPrtUnit, for multi-serial preferences and more flexible printer-preferences.

OpenScreenTagList(), OpenScreen(), OpenWindowTagList(), OpenWindow(), and the internal BorderPatrol() routine now go through stack-swapping. This protects applications from increased stack usage in these calls.

The Preferences LaceWB field (and whether the pretend screen mode from GetScreenData() is lace or not) now solely depends on the height of the text overscan rectangle of the true mode of the Workbench. This helps older applications opening on modes such as double-NTSC 640x400.

Intuition now handles MakeVPort() failure. Intuition will blank any failed ViewPort, and attempt to remake it at each opportunity. MakeScreen(), RethinkDisplay(), and RemakeDisplay() now have return codes that reflect MakeVPort() failure.

Rendering Optimizations

Several important rendering optimizations make Intuition appear snappier and cleaner. Here is a partial list:

- When a WFLG_ACTIVATE window is opened, Intuition now activates it synchronously. The big benefit is that the window's border is no longer drawn inactive then activate.
- EasyRequests and AutoRequests used to consist of a window with a requester inside, which meant two layers. This consumed memory and slowed down requester and other window operations. Now, the gadgets and imagery are brought up directly in the window, saving a layer.
- The window sizing/dragging rubber-band box is now much faster.
- The bar-layer of each screen no longer does any backfill processing, since Intuition fully re-renders the screen bar anyway.
- Intuition now avoids spurious border and gadget refresh and sending a spurious IDCMP_REFRESHWINDOW event. Formerly, if an application had not cleared its window's damage when another damage-causing operation (e.g., menus, window sizing/movement) occurred on the same screen, another round of refresh was performed. Intuition now can tell if the window's layer had been damaged since the last IDCMP_REFRESHWINDOW message went out.

- Menus are brought on-screen somewhat faster than before, and are removed very much faster than before. (3.01 and up only)

- A lot of work has been done to reduce window border and gadget flashing during window resize operations. (3.01 and up only)

Bug Fixes

NextPubScreen() could write a zero-byte one byte past the end of the buffer the caller supplies. It no longer does this.

Clicking in the no-window area of a screen now makes that screen the active screen (for purposes of autoscrolling).

Fixed a long-standing bug where REQCLEAR messages weren't being sent when a requester having no layer is taken down while other requesters are still up in the window.

Setting a negative minimum width or height with WindowLimits() no longer allows you to crash the computer by turning a window inside out.

SetWindowTitles() now properly erases remnants of the previous title, even when odd extenders happen.

A ghosted string gadget no longer causes patterning in string gadgets that precede it in the list and have a non-zero container background-pen.

NewModifyProp() of a disabled prop gadget no longer clears away part of the ghosting.

SetGadgetAttrs() to a proportional gadget no longer can cause the window's installed clip-region to be lost.

The mouse-pointer no longer blanks in the first gap when three interlaced screens up.

Several causes of sprite-pointer jumping have been fixed.

The ROM default sprite pointer now is the 2.0/3.0 one, instead of the 1.3 one.

DClips of coerced ViewPorts are finally scaled as correctly as possible.

If you used a boopsi string gadget as an integer gadget, with Intuition supplying the buffer,

and you specified a STRINGA_MaxChars of > 15, you would get a mismatched FreeMem() when the gadget is disposed. This is now fixed.

There is a bug in 3.0 (not in 2.0x and fixed in 3.01) where the autoscroll boundary was inadvertently switched to be the DClip of the active screen, where it used to be the "hull" of the DClips of all the screens. If there are two screens in the system with different DClips, the mouse can be way outside the DClip of the smaller screen. If that screen is active, it will AutoScroll at a ridiculous rate. For example, if the mouse is seventeen pixels below its DClip, moving it down one pixel causes the screen to autoscroll by eighteen, instead of one. This is now fixed.

Starting with 3.01, Intuition now updates its internal time values based on (nearly) *any* InputEvent it receives, instead of just IECLASS_TIMER ones. The problem was that outgoing IntuiMessages get their time from this internal time, which meant that IntuiMessage time was the time-stamp of the most recent timer tick, instead of the time-stamp of the event that actually triggered this IntuiMessage. This problem completely precludes correlating an IntuiMessage with the InputEvent that caused it, which is important for tablet people, for example. A SetPatch for earlier ROM versions is being considered.

In 3.00, there is a bug where the part of a window obscuring the title-bar area of a SCREENQUIET screen wasn't erased when the window was closed or moved away. Effectively, Intuition was relying on a layers side-effect that was optimized out for V39. Intuition fixes this for 3.01.

Starting with 3.01, when Intuition splits a single InputEvent into button and movement components, the button event is now sent first. This fixes some inconsistencies with extended input information like pressure, as well as odd behavior of the qualifiers, in particular IEQUALIFIER_MIDBUTTON.

◆

Changes in the V39 OS

by Martin Taillefer

Many changes were made to the system software for V39. The major differences are in the graphics subsystem and in Intuition. There were also hundreds of changes made to other areas of the system software. This document presents the majority of these changes, and explains the possible impact of these new features on the developer.

V39 requires both a new disk set, and a new ROM. V38 is a disk-based upgrade requiring only a new disk set. For a complete list of changes in V38, refer to "Release 2.1 Overview" in the *V39 Release Notes*, available from CATS (part number: V3.0).

The main changes to the contents of the disks in V39 are detailed in Appendix A and in Appendix B. Here are the highlights of changes to system disk organization:

- The V38/V39 system software is shipped on FFS floppies. This is mainly to increase available storage.

- Under V38, a Storage drawer was created on the Extras disk. Under V39, a separate Storage disk is provided. Storage contains the same five drawers as in the Devs drawer. This is where things that are not currently used are kept. To activate a monitor, the user is expected to drag the desired monitor icon from Storage/Monitors to Devs/Monitors. Same applies for printer drivers, keymaps, DOS drivers, and datatypes

- The Fonts disk is now called FONTS instead of AmigaFonts. This enables the disk to be used directly whenever fonts are needed by the system, instead of requiring the user to assign FONTS: to the disk.

- Note that there are different specific disk sets shipped with different machines and packages. For example, low-end V38 systems do not get a Locale disk, and instead have a Locale directory on the Workbench disk.

- Deleted Files. The following system files present under V37 are no longer included with the system software.

Devs/narrator.device	This device is deleted as part of the removal of speech support.
Libs/translator.library	This library is deleted as part of the removal of speech support.

Tools/Colors	This tool has been removed in V39 because its functionality conflicted with pen sharing, and was generally not very nice to applications.
Utilities/Display	This program is replaced under V39 by the more flexible and powerful MultiView utility.
Utilities/More.info	This icon has been removed because we want to encourage users to use MultiView instead of More when running from Workbench. The More program itself remains on the disk since it is referenced by many read me files, and the like.

- Speech Support. The various components supporting synthesized speech are no longer included with the operating system. This can be a serious liability for applications depending on synthesized speech for correct operation. The V37 components supporting speech still function correctly under V38 and V39. The V38/V39 installation procedures do not remove the V37 files when updating a system.

Further etails of how the system disks have changed can be found in Appendix A and B. V39 ROM changes are listed in Appendix C and D.

V38/V39 API Changes

This section discusses what changes in the programming model can affect developers.

Finding Version Information

An important point to mention is how to determine if a system is running V38 instead of V37. The recommended approach is to open version.library, and check its version. For example:

```
struct Library *VersionBase;
if (VersionBase = OpenLibrary("version.library",0))
{
       if (VersionBase->lib_Version >= 38)
       {           /* user is running at least V38 Workbench */       }
       else
       {           /* user is running at most V37 Workbench */        }
}
else
{     /* can't tell what the user is running, assume the minimum version
       * that your application supports        */
}
```

The above technique lets you determine which general version is in use for the disk-based software. *Never* assume this is a reflection of every other library in the system. For example, if you need features that are only present in V38 asl.library, you must explicitly check the version of asl.library before using it. The same is true for all other system libraries.

To determine the general version of the ROM, use SysBase->LibNode.lib_Version.

GadTools Library

Many enhancements were made to GadTools for V39. Below you will find short descriptions of some of the new features. A complete list of all the new features is provided in Appendix A, and more details can be had in gadtools.doc.

Menus. GadTools fully supports Intuition's NewLook menus. NewLook menus can be added to an application by providing the {WA_NewLookMenus, TRUE} tag to OpenWindowTags(), and providing the {GTMN_NewLookMenus, TRUE} tag to LayoutMenus(). These two tags will give your application NewLook menus. The pens used to render these menus are controllable by the user through the V39 Palette prefs editor.

You can now put an arbitrary command string in the right-hand side of a menu, where the Amiga-key equivalent normally goes. To do this, point the NewMenu.nm_CommKey field at the string (e.g., "Shift-Alt-F1), and set the new NM_COMMANDSTRING flag in NewMenu.nm_Flags.

GT_GetGadgetAttrs(). GadTools now has a GT_GetGadgetAttrsA() function which retrieves attributes of the specified gadget, according to the attributes chosen in the tag list. For each entry in the tag list, ti_Tag identifies the attribute, and ti_Data is a pointer to the long variable where you wish the result to be stored.
Here is an example:

```
LONG top = 0;
LONG selected = 0;
LONG result;
result = GT_GetGadgetAttrs(listview_gad, win, NULL,
                           GTLV_Top, &top,
                           GTLV_Selected, &selected,
                           TAG_DONE);
if (result != 2)
{
    printf( "Something's wrong!" );
}
```

Palettes. GadTools palette gadgets got a major overhaul for V39. There are a few new features, some reduction in memory usage, and an increase in performance.

Palette gadgets no longer display a box filled with the selected color. The selected color is instead denoted by a box drawn around the color square in the main palette area. This change slightly impacts the size of palette gadgets. Depending on the conditions, V39 PALETTE_KIND gadgets can be slightly smaller than the V37 ones.

Another change which can affect the size of the palette gadgets is the smarter layout of the color squares. An attempt is now made to keep the color squares as square as possible, based on the aspect ratio information obtained from the gfx database. As many color squares as possible are put on the screen, until things get too small in which case the upper colors are thrown away.

The GTPA_ColorTable tag was added in support of sparce color tables. It lets you define arbitrary pens to be used to fill-in the color squares of the palette gadget.

The GTPA_NumColors tag lets you define the exact number of colors to display in the palette gadget. Under V37, the GTPA_Depth tag played the same role. The problem with GTPA_Depth is that only multiples of 2 in number of colors can be specified. GTPA_NumColors allows any number of colors.

Listviews. GadTools listviews got a major overhaul for V39. There are a few new features, some reduction in memory usage, and an increase in performance.

The biggest difference that is readily noticeable is the disappearance of the display box at the bottom of the scrolling list area. Instead of showing the selected item in the display box, the selected item remains highlighted within the scrolling list. This was done to prepare listviews for multi-select support. Multi-selection requires a highlighting scheme as a display box would not work. The removal of the display box has a slight impact on the size of the listview. Listviews that had a display box under V37 got slightly smaller vertically under V39.

The other major addition to GadTools listviews is GTLV_CallBack. This tag allows a callback hook to be provided to gadtools for listview handling. Currently, the hook only gets called to render an individual item. Every time GadTools wishes to render an item in a listview, it invokes the call back function, which can do custom rendering. This allows GadTools listviews to be used to scroll complex items such as graphics and images.

Finally, listviews can now be disabled using {GA_Disabled, TRUE}.

The ASL Library

ASL got rewritten for V38. Its file requester is now extremely fast and offers more features to the user, and to the programmer. Its font requester now comes up about twice as fast the first time it is displayed, and instantaneously for subsequent uses. ASL also now includes a screen mode requester, which is very useful in the context of AA, and will be invaluable for AAA and RTG.

New Names. V38 *<libraries/asl.h>* contains new names for most of the structures, tags, and flag bits used by ASL. The names are now consistent, and more descriptive. We encourage the use of the new names as much as possible.

To ensure your code only uses the new names, just define the symbol ASL_V38_NAMES_ ONLY before including *<libraries/asl.h>*:

```
#define ASL_V38_NAMES_ONLY
#include <libraries/asl.h>
```

The symbol definition will prevent the old-style names from being defined. This is the same method used to disable the pre-V37 names in Intuition header files.

Don't Trust Me. The many filtering tags that you can supply to the ASL requesters are only advisory in nature. For example, even if you request that only non-proportional fonts be allowed in the font requester, the user can still type in the name of a proportional font. The user can also enter out-of-bounds values for any numeric parameter. You must check all results for sanity prior to using them.

Common ASL Tags. The tags for the different requesters are now distinct from one another instead of being all grouped together. All options that could be set via flag bits now have individual tags to control them. For example, there is now an ASLFR_DoSaveMode tag that does the same as the FRF_DOSAVEMODE flag bit.

Some basic tags are supported by all requester types:

ASLXX_Window. This indicates the parent window of the requester. You provide a window pointer, and if you don't specify an ASLXX_Screen tag, then the ASL requester will open on the same screen as this window.

ASLXX_PubScreenName. You provide this tag with the name of a public screen to open the ASL requester on.

ASLXX_Screen. You provide this tag with a pointer to a Screen to open on. If none of the above three tags are specified, then the ASL requester will open on the current default public screen. This will most likely by Workbench.

ASLXX_PrivateIDCMP. By default, ASL requesters use the same IDCMP port as their parent window (specified using ASLXX_Window). You can request that a private IDCMP port be used by setting this tag to TRUE. If you don't provide a parent window, then a private port is automatically used and this tag need not be provided.

ASLXX_IntuiMsgFunc. You provide this tag a pointer to a Hook structure, which indicates a routine to run whenever an IntuiMessage arrives at the ASL requester's IDCMP port that is not meant for the requester's window. This happens whenever the ASL requester is sharing the parent's window IDCMP port, and a message for the parent window arrives at the port. The function being called can process the message for the application.

ASLXX_SleepWindow. If you set this tag to TRUE, ASL will put the parent window (specified using ASLXX_Window) to sleep until the ASL requester is satisfied. This involves blocking all input in the parent window, and displaying a busy pointer in it.

ASLXX_TextAttr. You can provide a pointer to a standard TextAttr structure which defines

the font to use in the ASL requester. If this tag is not supplied, the default behavior is to use the font of the screen on which the requester opens. Note that just like for Intuition objects, the font indicated by the TextAttr structure must already be in memory before using the requester. There is no guarantee that ASL will actually use the font you asked for. If that font is too large, ASL will use a smaller one. Also, as of V39, the file requester's file list requires a monospace font. So if you provide a proportional font, the file requester will use the system default font instead for its list of files.

ASLXX_Locale. You pass this tag a pointer to a Locale structure, as obtained from the locale.library/OpenLocale() function. The locale is used to determine in which language, and using which country information, the various requesters should be displayed. If this tag is not provided, or its value is NULL, then the default system Locale is used. This corresponds to what the user has currently picked in the Locale prefs editor. As of V39, certain items such as the dates in the file requester always use the system default Locale instead of the Locale provided with this tag.

ASLXX_TitleText. Provide this tag a string which will be used for the title of the requester. If this tag is not provided, the requester has no title.

ASLXX_PositiveText. Provide this tag a string which will be used for the label of the positive choice gadget in the requester. If this string is not provided, a localized default is used (English default is "OK"). We recommend using this tag to make the action of the various requesters more clear to the user. For example, when a file requester is being displayed to load a file, it is clearer if the gadget says "Open" instead of simply "OK".

ASLXX_NegativeText. Provide this tag a string which will be used for the label of the negative choice gadget in the requester. If this string is not provided, a localized default is used (English default is "Cancel").

ASLXX_InitialLeftEdge, ASLXX_InitialTopEdge, ASLXX_InitialWidth,

ASLXX_InitialHeight. These four tags let you specify the position and size of the ASL requester window. These are only requests that ASL may decide to ignore. When an ASL requester is closed, it is easy to determine the position and size of the requester when the user closed it. It is a good idea for an application to remember these values and use them in subsequent invocations of the requester.

The File Requester. The file requester supports a number of tags. Some of the tags introduced in V38 are discussed here.

Note that the future holds many changes in the ASL file requester that will substantially increase its flexibility to both the programmer and the user. If you use the ASL file requester in your applications today, you will automatically get the majority of these new features when they become available. Rolling your own file requester means a lot of extra work, and means that you will not automatically benefit from the new interface when it becomes available.

ASLFR_DoSaveMode. Setting this tag to TRUE indicates the file requester is being used for saving information. When in save mode, the file requester changes slightly. The most obvious change is the use of a different set of colors in the file list. This is a direct queue to the user that something is being selected for writing, not reading. Second change is that if the user enters a directory name which doesn't exist, ASL will prompt the user to see if the user wishes to create the directory. This lets the user create new directories to save files into. There are likely to be more differences between normal and save mode in the future. Please use this tag when appropriate.

ASLFR_RejectIcons. When set to TRUE, this tag prevents icons (.info files) from being displayed in the file requester. If this tag is not specified, icons will be displayed by the file requester. Please use this tag in all your software. Workbench users should never have to see .info files. The default behavior of the file requester is to display .info files, which is incorrect. Unfortunately, this default behavior cannot be changed due to compatibility.

ASLFR_DoPatterns. Setting this tag to TRUE causes a Pattern gadget to be displayed in the file requester, which allows the user to enter AmigaDOS patterns to filter out files. The default is to have no pattern gadget.

ASLFR_DrawersOnly. Setting this tag to TRUE causes the file requester to have no File gadget, and to only display directory names in its file list. This is a useful option if you wish to have the user select a destination directory for a particular task.

ASLFR_RejectPattern. Provide an AmigaDOS pattern to this tag, and any files matching this pattern will not be displayed in the file requester. This pattern can never be edited by the user. Note that the pattern you provide here must have already been parsed by dos.library/ParsePatternNoCase().

ASLFR_AcceptPattern. Provide an AmigaDOS pattern to this tag, and only files matching this pattern will be displayed in the file requester. This pattern can never be edited by the user. Note that the pattern you provide here must have already been parsed by dos.library/ParsePatternNoCase().

ASLFR_FilterDrawers. Setting this tag to TRUE causes the ASLFR_RejectPattern, ASLFR_AcceptPattern, and the Pattern text gadget to also apply to filter drawer names. Drawers are normally never affected by pattern filtering.

The Screen Mode Requester. The screen mode requester allows the user to select amongst the wide range of display modes available on the Amiga. Future chip sets, and the RTG effort will yield a very large additional number of display modes. Using the ASL requester is a good way to ensure that you benefit from the maximum upwards compatibility possible. In many cases, by using the screen mode requester, your applications will automatically be able to open displays in new modes that are introduced in future OS revisions.

An important point to note is that the interface presented to the user by the screen mode requester is likely to change dramatically in the future. This is because the increased number of available modes will make the current interface difficult to use and understand at best. A new scheme is being developed that should allow the user to more easily and accurately pick display modes. If you use the ASL screen mode requester in your applications instead of rolling your own version, your application will automatically get this new interface when it becomes available.

The ultimate goal of the screen mode requester is to return a graphics mode id. The mode ids are used since V37 to distinguish between the different display modes in the system. The screen mode requester can also return additional information including requested display sizes and depth.

As all other ASL requesters, you get the result of a request by reading the contents of the requester structure after the AslRequest() calls return success. There are two fields worth mentioning in the ScreenModeRequester structure: sm_BitMapWidth and sm_BitMapHeight. These values define the values to pass to AllocRaster() and InitBitMap() when hand-building a BitMap structure to display the requested width, height, and depth. This is useful when running on a V37 ROM. For V39 use, simply use the values in sm_Width and sm_Height, and pass those to AllocBitMap() directly.

Here are the most important screen mode requester tags.

ASLSM_InitialDisplayID. This tag sets the initial mode that is selected in the mode listview. You provide it a graphics mode id, and that mode will be initially selected in the file requester.

ASLSM_InitialDisplayWidth, ASLSM_InitialDisplayHeight, ASLSM_InitialDisplayDepth. These tags determine the initial setting of three gadgets that can optionally be displayed in the requester. The default if these tags are not provided are a width of 640, a height of 200, and a depth of 2.

ASLSM_InitialOverscanType. This tag determines the initial setting of the Overscan Type cycle gadget. This optional gadget lets the user pick which overscan setting to use for the given mode. The values you can provide to this tag are: OSCAN_TEXT, OSCAN_STANDARD, OSCAN_MAXIMUM, and OSCAN_VIDEO. OSCAN_VIDEO is only available starting with V39, it is not in V38. The four OSCAN_XX constants are defined in *<intuition/screens.h>*.

ASLSM_InitialAutoScroll. This tag can be set to TRUE or FALSE and determines what the initial state of the option AutoScroll checkbox gadget should be.

ASLSM_DoWidth, ASLSM_DoHeight, ASLSM_DoDepth, ASLSM_DoOverscanType, ASLSM_DoAutoScroll. These tags control which of the optional gadgets is displayed in the screen mode requester. The default is to have none of these present. Setting any of these tags to TRUE will make the gadget appear.

ASLSM_MinWidth, ASLSM_MaxWidth, ASLSM_MinHeight, ASLSM_MaxHeight, ASLSM_MinDepth, ASLSM_MaxDepth. These tags let you specify limits to the values the user is allowed to enter in the requester.

ASLSM_PropertyFlags, ASLSM_PropertyMask. These two tags cooperate to allow a flexible means of filtering out unwanted display modes. Each display mode in the graphics database has a given set of properties associated with it. These tags allow you to determine which modes to display in the mode list of the requester based on the properties of the modes.

The mode properties are defined as a ULONG of flag bits. The definitions for all supported properties are in <*graphics/displayinfo.h*>. Examples include DIPF_IS_WB and DIPF_IS_LACE.

ASLSM_PropertyMask defines which properties you are concerned with. Those are the only bits for which the specific setting matters to you. The setting of all other property bits doesn't matter to you.

ASLSM_PropertyFlags defines exactly in which state the property flags you are have shown interest in via the ASLSM_PropertyMask tag should be.

The way ASLSM_PropertyMask and ASLSM_PropertyFlags interact is indentical to how the two flags parameters interact in exec.library/SetSignal(). This is how ASL uses the tag values internally:

```
if ((displayInfo.PropertyFlags & propertyMask) ==
    (propertyFlags & propertyMask))
{
    /* Mode accepted */
}
else
{
    /* Mode rejected */
}
```

An example can help understand the relationship between the two tags. If you want to display only screen modes that can be used for the Workbench screen, and that are not interlaced, you would set ASLSM_PropertyMask to (DIPF_IS_WBI DIPF_IS_LACE), and ASLSM_PropertyFlags to (DIPF_IS_WB). The mask value means you are interested in the state of the DIPF_IS_WB and DIPF_IS_LACE bits. The flags value says you want the DIPF_IS_WB to be set, and the DIPF_IS_LACE bit to be clear.

The Locale Library

The locale.library provides the core of system localization. It is explained in detail in the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0). Also refer to locale.doc for more information.

The Bullet Library

The bullet.library contains the Compugraphic outline font rendering engine. It is explained in the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0). Also refer to bullet.doc for more information.

Color Wheel and Gradient Slider

The color wheel and gradient slider are two new BOOPSI classes introduced in V39. Together, they offer a very nice interface to let the user select or adjust colors. The V39 Palette prefs uses both classes in its interface.

Using either of these classes requires you to first open them as libraries, and then create new BOOPSI objects using the NewObject() function:

```
if (ColorWheelBase = OpenLibrary("gadgets/colorwheel.gadget",0))
{
    cw = NewObject(...);
}
```

For an introduction to programming the color wheel and gradient slider refer to the *Amiga Mail* newsletter, Sept/Oct 1992 issue, page IV-91 also published in *V39 Release Notes* available from Commodore's CATS department (part number: V3.0).

The colorwheel.gadget. The color wheel class provides the ability to create gadgets enabling the user to control the hue and saturation components of an HSB (Hue-Saturation-Brightness) color space. The companion gradient slider class enables control of the brightness component of the color space.

The color wheel can operate on screens of any depth, and adapts its rendering to the number of colors available. The system's pen sharing system is used in order to maximize the number of colors used by the wheel. A color wheel gadget is (normally) responsible for choosing it's own color pens to draw in (using graphics.library/ObtainBestPen()). However, the creator of the gadget can "donate" some of its pens to the gadget, using the WHEEL_Donation tag.

The reason that the color wheel picks its own colors is because it has the ability to display several different layouts depending on the number and variety of colors available. For example, when opening on a screen of low depth or when opening on a screen where all the

pens have already been allocated exclusively, the gadget will display a "monochrome" version of the color wheel, where instead of colored segments, the letters "R" (for red), "G" (for green), "B" (for blue), "Y" (for yellow), "C" (for cyan) and "M" (for magenta) will be used as labels.

You can talk to the color wheel using HSB or RGB, even though the color wheel only really deals with HSB in its user-interface. All communications with applications are performed with full 32-bit color component values. Internally everything is currently kept and processed in 16-bit space, although this might change in the future.

Here are the main tags supported by the color wheel class:

WHEEL_Hue. Lets you control the Hue component of the color wheel. This is effectively the angle around the wheel where the desired color lies. A hue value of 0 is all red, and nothing but red. Increasing the value moves the color towards all green at $55555555, full blue at $AAAAAAAA, and back to red at $FFFFFFFF.

WHEEL_Saturation. Lets you control the Saturation component of the color wheel. This is effectively the distance from the center of the wheel where the desired color lies. A saturation value of 0 puts the knob at the center of the wheel and always yields white. Increasing the value progressively moves the knob farther away from the center. until the value $FFFFFFFF is reached in which case the knob is as far as it can go.

WHEEL_Brightness. Lets you control the Brightness component of the color wheel. The color wheel does not itself have any means of displaying or editing brightness, but it does maintain this value internally. Used with the WHEEL_GradientSlider tag, you can control the value of a gradientslider object by passing WHEEL_Brightness to a color wheel. A brightness value of 0 means all black. Increasing the value progressively brightens the current color, until the value $FFFFFFFF is reached in which case the color is as bright as it gets.

WHEEL_Red, WHEEL_Green, WHEEL_Blue. Let you specify new RGB values for the color wheel. The values provided are converted into HSB values are then used.

WHEEL_Screen. This is a required tag that lets you specify the screen on which the color wheel is to be displayed. Note that once a color wheel is created, the screen should not be closed until the color wheel object is discarded using DisposeObject().

WHEEL_BevelBox. If you set this tag to TRUE, a bevel box will be drawn around the color wheel object.

WHEEL_GradientSlider. This tag lets you attach a gradient slider object to the color wheel. You give this tag a pointer to a gradient slider object obtained previously from NewObject(). Once this is done, anytime the various tags that can affect the brightness component of the current color is sent to the color wheel, the color wheel automatically

changes the value of the attached gradient slider to match that new brightness value. Reading the brightness value from the color wheel returns the current value indicated by the gradient slider.

Using this tag effectively allows you to treat the color wheel and gradient sliders as a single gadget. Once things are set up, all communications occur through the wheel object, and the gradient slider can pretty much be ignored.

The gradientslider.gadget. The gradient slider gadget class is a type of non-proportional slider. The primary feature of the gradient slider is it's appearance. Unlike normal sliders, a gradient slider can display a "spread of colors" or "color gradient" in the slider container box. The "knob" or "thumb" of the slider appears to slide on top of this color gradient.

The color gradient effect is built-up using a combination of multiple pens and half-tone dithering. The application must tell the slider exactly which pens to use in creating the gradient effect, and in what order to use them. Essentially, it does this by passing an array of pens (terminated by ~0, just like a PenSpec) to the slider. The first pen in the array is used as the color at the top of the slider (or left, if it is horizontal), and the last color in the array is used at the bottom (or right). The other pens will be used at evenly spaced intervals in between. Dithering is used to smoothly fade between the pens, allowing the illusion of a continuous change in color.

Here are the main tags supported by the gradient slider class:

GRAD_MaxVal. Lets you set the maximum value supported by the slider. This must be in the range $0..$FFFF.

GRAD_CurVal. Lets you set the current value of the slider. This should be in the range 0..GRAD_MaxVal.

GRAD_PenArray. Lets you specify an array of pens that the slider should use to create its gradient background. The array can contain any number of pens, and is terminated with a pen value of ~0. These pens can be allocated as shared, since their RGB value is not altered by the slider. The first pen is used on the top or left of the slider, and the last pen is used on the bottom or right. All other pens are evenly spaced out and used in between. Dithering is used between the pens to enhance the smoothness of the gradient transition.

A NULL pen array causes the background of the slider to be rendered in the screen's background color. A pen array containing only a single pen causes the background to be rendered using that pen.

New File Systems

The V39 ROM file system supports 6 different disk formats:

- ❑ DOS\0 This is the traditional Amiga file system. It has 488 bytes of data per block. It offers a high level of redundancy and validation which makes it very reliable. It is also fairly slow.

- ❑ DOS\1 This is the original Fast File System introduced in 1.3. It uses a disk layout quite similar to DOS\0, except it forgoes some redundancy in the name of speed. It stores 512 bytes per block, and runs substantially faster than DOS\0.

- ❑ DOS\2 This is an international flavor of DOS\0. The only difference between this format and DOS\0 is the hashing algorithm used to locate the files and directories on the disk. DOS\0 has a bug in dealing with making international characters case-sensitive. DOS\2 corrects this bug, and is otherwise identical to DOS\0.

- ❑ DOS\3 This is an international flavor of DOS\1. The only difference between this format and DOS\1 is the hashing algorithm used to locate the files and directories on the disk. DOS\1 has a bug in dealing with making international characters case-sensitive. DOS\3 corrects this bug, and is otherwise identical to DOS\1.

- ❑ DOS\4 This is a directory-caching version of DOS\2. It has the same general disk layout, with the addition of special directory cache blocks. These cache blocks store the contents of each directory. The advantage of this is that getting a directory listing can be an order of magnitude faster than on normal DOS\2 disks. The directory caches require a slight amount of disk space, and maintaining them slightly slows down file creation, deletion, and update. This file system is mostly meant for floppies, as it doesn't generate a very noticeable performance increase on hard drives.

- ❑ DOS\5 This is a directory-caching version of DOS\3. It has the same general disk layout, with the addition of special directory cache blocks. These cache blocks store the contents of each directory. The advantage of this is that getting a directory listing can be an order of magnitude faster than on normal DOS\3 disks. The directory caches require a slight amount of disk space, and maintaining them slightly slows down file creation, deletion, and update. This file system is mostly meant for floppies, as it doesn't generate a very noticeable performance increase on hard drives.

Disk Block Format. Here are some notes concerning changes to the disk block format required by the new SetOwner() call, and more importantly the changes present in DOS\4 and DOS\5 to support directory caching.

The format of root blocks is unchanged under V39.

The format of file header blocks is slightly different in order to store owner information. One of the reserved fields is now being used:

```
struct FileHeaderBlock
{
    ULONG  fhb_Type;          // T.SHORT
    ULONG  fhb_OwnKey;          // key of this block
    ULONG  fhb_HighSeq;       // total blocks in file
    ULONG  fhb_DataSize;      // number of data block slots used
    ULONG  fhb_FirstBlock;    // first block in this file
    ULONG  fhb_Checksum;      // checksum of this block

    ULONG  fhb_HashTable[];  // variable number of hash table entries

    ULONG  fhb_Reserved       // always 0
    ULONG  fhb_OwnerXID       // owner UID/GID
    ULONG  fhb_Protect        // protection bits
    LONG   fhb_ByteSize       // total size of file in bytes
    char   fhb_Comment[92];   // comment as a BCPL string
    ULONG  fhb_Days;          // creation date and time
    ULONG  fhb_Mins;
    ULOGN  fhb_Ticks;
    char   fhb_FileName[36]; // filename as BCPL string
    ULONG  fhb_Link;          // pointer to object header is linked to
    ULONG  fhb_BackLink;      // pointer to previous object in link chain
    ULONG  fhb_HashChain;     // next entry with same hash value
    ULONG  fhb_Parent;        // pointer back to parent directory
    ULONG  fhb_Extension;     // 0 or pointer to first extension block
    ULONG  fhb_SecType;       // ST.FILE
};
```

User directory blocks were modified to add the same owner information as for file header blocks. Under DOS\4 and DOS\5, the user directory blocks also point to the new directory cache blocks.

```
struct UserDirectoryBlock
{
    ULONG  udb_Type;          // T.SHORT
    ULONG  udb_OwnKey;        // key of this block
    ULONG  udb_SeqNum;        // highest seq (always 0)
    ULONG  udb_DataSize;      // always 0
    ULONG  udb_NextBlock;     // always 0
    ULONG  udb_Checksum;      // checksum of this block

    ULONG  udb_HashTable[];  // variable number of hash table entries

    ULONG  udb_Reserved;      // always 0
    ULONG  udb_OwnerXID;      // owner UID/GID
    ULONG  udb_Protect;       // protection bits
    ULONG  udb_Junk;          // not used (always 0)
    char   udb_Comment[92];   // comment as a BCPL string
    ULONG  udb_Days;          // creation date and time
    ULONG  udb_Mins;
    ULONG  udb_Ticks;
    char   udb_DirName[36];  // name as a BCPL string
    ULONG  udb_Link;          // pointer (key number) to object linked to
    ULONG  udb_BackLink;      // back pointer to previous hard link header
    ULONG  udb_HashChain;     // next entry with same hash value
```

```
    ULONG  udb_Parent;       // pointer back to parent
    ULONG  udb_DirList;      // DOS\4 and DOS\5 use this for the dir cache
    ULONG  udb_SecType;      // ST.USERDIR
};
```

A new block type added in support of DOS\4 and DOS\5 is the file list block. This is where directory information is cached by the file system. When looking at a file list block, is it critical to check the fhl_Type field of the block to ensure that it is a known block format. Unknown block formats should be ignored. On a DOS\4 and DOS\5 partitions, the file system is able to rebuild cache blocks if it finds older versions present, or if it finds no cache block for a given directory.

```
struct FileListBlock
{
    ULONG  fhl_Type;        // T.DIRLIST
    ULONG  fhl_OwnKey;      // key of this block
    ULONG  fhl_Parent;      // key of parent directory
    ULONG  fhl_NumEntries; // amount of data on this block
    ULONG  fhl_NextBlock;  // next block in sequence of file list blocks
    ULONG  fhl_Checksum;   // checksum of this block
};
```

Following the above structure on a disk block comes the actual cache information for the directory. There is one cache entry for every file and directory within the cache. These entries vary in size based on the length of the file name and comment for the entry. A cache entry looks like:

```
struct ListEntry
{
    ULONG fle_Key;         // Key (fhb block) of file
    ULONG fle_Size;        // size in bytes
    ULONG fle_Protection; // protection bits
    ULONG fle_OwnerXID;    // owner info
    UWORD fle_Days;        // date (allows 179 years)
    UWORD fle_Min;         // 0..(60*24 - 1)
    UWORD fle_Tick;        // 0..3599
    UBYTE fle_Type;        // ST.XXXXX (all fit in a byte)
    char  fle_FileName[]; // variable length BCPL string
    char  fle_Comment[];  // variable length BCPL string
};
```

These are the values for the various constants referred to above:

```
// block type constants
#define T.DELETED 1
#define T.SHORT   2
#define T.LONG    4
#define T.DATA    8
#define T.LIST    16

DCFS_REV      1
DIRLIST_KEY   32
DIRLIST_MASK  0xfffffffe0
T.DIRLIST     (DIRLIST_KEY | DCFS_REV)

ST.FILE      -3
ST.ROOT       1
ST.USERDIR   2
```

DOS Workarounds

V37 dos.library has a few important bugs. These bugs are fixed in V39, and it is easy to work around many of these bugs under V37 to make sure you don't get into trouble.

AttemptLockDosList(). Under V37, AttemptLockDosList() would sometimes return 1 on failure instead of NULL. To check for failure of AttemptLockDosList(), simply use:

```
dl = AttemptLockDosList();
if (!dl || (dl == (struct DosList *)1))
    /* failed */
```

FGets(). Under V37, FGets() will overrun the buffer you give it by one byte, if it doesn't encounter a line feed or EOF before filling up your buffer. The work around is very simple: pass buffersize-1 to FGets().

ParsePatternNoCase(). Until V39, ParsePatternNoCase() did not correctly treat character classes as case-insensitive. That is [a-z] was different than [A-Z]. The work around for this bug is to simply convert all characters of the pattern to upper case before passing it to ParsePatternNoCase(). Just use the utility.library/ToUpper() function to convert all characters of the pattern to upper case.

SetVBuf(). Until V39, the SetVBuf() function didn't do anything. Starting with V39, it actually does in fact set the buffer sizes. If you use any of the buffered DOS IO calls, you can get a noticeable increase in performance by increasing the buffer size of your files. A good size is 4K. You can simply add a SetVBuf() call after opening each of your files. It won't do anything under V37, and will work under V39. In most cases, it is not even necessary to check for failure of SetVBuf(). When the call fails, it just leaves the file handle with the same buffering it had previously. That's fine, you just won't get the performance increase for that run of the program.

SetVBuf() was not activated in the initial release of 3.0, it only becomes available with 3.01.

Commodities Library

The commodities.library underwent a major overhaul in V38. The result is a much smaller, much less CPU hungry, and much less memory hungry library. Many bugs were also found and fixed. Finally, a few new features were included. All of the commodity programs shipped with the system were also revised.

ReadArgs(). Commodities now use ReadArgs() instead of the custom command-line parsing routines in amiga.lib. This makes them more consistent with the rest of the system. We encourage third party commodity writers to also switch to ReadArgs().

Keyword Synonyms. ParseIX() now understands many synonyms for keys and qualifiers. The intent was to provide more consistent naming, and to decrease the likelihood the user would make a mistake while entering a key sequence. For a complete list of all the keywords and synonyms refer to the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0)

Input Events. A bug that was found in a few applications during V39 development was the incorrect initialization of InputEvents inserted into the input.device's input stream. These events can work most of the time, and suddenly not work with specific applications. Of specific interest, the inputEvent->ie_SubClass was found to often contain garbage. This field should generally be set to 0. Another field that is often incorrectly initialized is the ie_TimeStamp field. Make sure this field has something meaningful in it. Note that using the IND_WRITEEVENT input.device command automatically initializes the ie_TimeStamp field.

Bugs In commodities.library. Under V37, input events output by translator objects were inserted into the input stream after the commodities.library input handler. This was contrary to the documentation. Starting with V38, events generated are inserted immediately following the translator object. This means commodity objects coming after the translator now get to see the translated events, while in V37, they wouldn't.

Another bug with translator objects is that they insert their attached list of input events in reverse order. This cannot be fixed due to compatibility problems. It is easily worked around by arranging the input events in the reverse order of how you want them to be inserted. The InvertString() function in amiga.lib generates an inverted list of input events, which is just fine for translator objects.

Starting with V39, input events generated by translator objects are stamped with the same time as the input event that activated the translator. This helps applications that look at the time of input events.

In V38, the ParseIX() function has a bug which prevents the following keys on the numeric keypad from being used in commodities: . 9 ( ) / * +. This bug is fixed in V39.

Until V39, input events of type IECLASS_NEWPOINTERPOS never got their extended tag list data copied when it needed to be. This can cause general confusion or crashes if a sender object is used on an event of that type. The only solution is to refrain from sending messages of type IECLASS_NEWPOINTER.

Until V39, the InvertKeyMap() function never initialized the ie_SubClass and ie_TimeStamp fields of the input event it created. The workaround to this bug is simple:

```
inputEvent->ie_SubClass           = 0;
inputEvent->ie_TimeStamp.tv_secs  = 0;
inputEvent->ie_TimeStamp.tv_micro = 0;
if (InvertKeyMap(ansiCode,inputEvent,NULL))
```

Mount Lists

The Mount procedure got a major overhaul in V38. The major change from the user's perspective is that the handlers that get mounted in the system can now be controlled exclusively through Workbench by dragging icons around. This is especially important in light of CrossDOS.

New File Format. Although the traditional DEVS:MountList file is still fully supported, a new method of storing mount information is now recommended. The new format involves simply splitting up the various entries that used to be in a mountlist, into separate files. Each file controls a unique handler. The format for the information in the mount files is the same as what is used in a traditional MountList with three exceptions. For example:

```
/* Aux-Handler mount entry */
Handler   = L:Aux-Handler
Stacksize = 1000
Priority  = 5
```

The differences with old style MountLists are as follows:

- Only a single device can be defined per file.
- The name of the device is not specified in the file and is instead the same as the name of the file.
- The # at the end of the entry can now be omitted.

Every parameter that can be specified in a mount file can also be specified in the tooltypes for the icon of the mount file. The parameters in the tooltyeps override any equivalent parameter setting present in the mount file. The format for the tooltypes is straightforward. For example:

```
LOWCYL=0
HIGHCYL=79
```

This lets the Workbench user easily control certain attributes associated with any given mount entry. For example, the WB user can easily set the size of RAD from tooltypes.

If the user wishes to only use your handler once in awhile, he can drag the icon for the handler in the Storage drawer. This prevents the handler from being mounted on every boot. If the user wants to use the handler for a given session, he can then simply double-click on the mount entry icon to get it mounted.

It is fairly simple to create an installation procedure that deals with both old style and new style mount files:

```
if (V38)
{
    /* Create a mount entry file.  If the user indicates he wants to use the
```

```
         * handler permanently, put it into DEVS:DOSDrivers.  Otherwise, just put
         * it into SYS:Storage/DOSDrivers.  Don't forget to provide an icon for
         * the mount entry.
         *
         * If the mount entry is in DEVS:DOSDrivers, it will get mounted
         * automatically when the system is rebooted.  The user can also mount
         * it for the current session by double-clicking on it.
         */
    }
    else /* V37 or before */
    {
        /* Create an old-style mount entry and append it to DEVS:MountList.
         * Add a Mount statement in the user-startup of the system.
         */
    }
```

File System Resource List. Mount now has the ability to fish out file systems from the FileSystem.resource list. When mounting a file system, Mount first look in the FileSystem.resource to see if the system already contains a file system of the correct DOS type. If there is such a file system, its seglist is used for the new handler.

For example, the CrossDOS file system registers itself in the FileSystem.resource the first time it is used. Any subsequent attempts to mount a handler having the same DOS type as the CrossDOS file system results in that handler using the same seglist as the original CrossDOS handler.

If a file system seglist is reentrant, it should be added to the FileSystem.resource list of file systems, which will enable it to be used in the manner described above. Be sure the seglist is totally reentrant before doing this (no global variables). Consult the Rom Kernal manual to learn how to add a file system to the FileSystem.resource list.

The new FORCELOAD keyword can be specified in mount entries. When its value is set to 1, it tells mount to always load the file system for this handler from disk, even if it is present in the FileSystem.resource list. This can be quite handy during the testing phase of a handler.

Unit Keyword. Stream handlers have always had the advantage of being able to let the user provide startup arguments to the handler via the Startup keyword. For example:

```
Startup = "this is a bunch of options"
```

The value of the parameter is available to the handler as a BSTR in the dol_Startup field of the handler DosList structure. This only works for stream handlers though, file system handlers make use of the dol_Startup field as a BPTR to the FileSysStartupMsg created for them by the Mount command.

The Unit keyword can now be used to passed textual options to a file system handler. If a string is specified in the mount entry, then the fssm_Unit field of the file system's FileSysStartupMsg will be a C pointer to the NULL-terminated argument string, instead of being the traditional unit number.

New Values For GlobVec. The GlobVec keyword in a mount list can now be set to -2 and -3 in addition to the other values already supported. Here is a table of possible values:

GlobVec Value	Meaning
0	Default global vector, used for handlers written in BCPL. Startup packet comes in a register.
-1	No global vector, used for C and ASM handlers. Process has to wait for startup packet.
-2	Same as -3, except for handlers written in BCPL.
-3	No global vector, used for C and ASM handlers. The difference with -1 is that the handler can do DOS IO prior to returning its startup packet. On the flip side, the handler is responsible for making sure there is only one instance of itself loaded by providing correct arbitration around the poking of the dol_Taskfield of the DosList structure. With a GlobVec of -1, this is taken care of by DOS.

Even though the values -2 and -3 are only allowed starting with the V39 Mount command, the V37 dos.library handles those values correctly. It's just that prior to V37, the Mount command would refuse to process these values.

Version Command

There has been some confusion about what the format of version strings are. The exact format is detailled in the *Amiga User Interface Style Guide* on page 110. The format is:

$VER: <name> <version>.<revision> (dd.mm.yy)

The string starts with $VER: followed by a space, followed by <name>. <name> is the name of the program. <name> can NOT contain any spaces. You can use underscores to achieve a similar effect.

After <name> comes a single space, followed by <version>. <version> is the major version number for the program. There should be no leading zeros.

After <version> comes a single space, followed by <revision>. <revision> is the minor version number for the program. There should be no leading zeros.

Following the revision number comes a space, and then the date. The date is specified in numeric form only, with no leading zeros on any of the components. First comes the day of the month, then the month, then the two digits year. In the future, a four digit year format will also be supported, but not yet.

Embedding a version string in the exact format described above will let the C/Version command find the version string. In the future, there will be a system call to enable applications, and other system tools, to obtain the version information from files easily.

ROM Tags. The version command not only looks for the $VER: version strings, it also search executable files for standard ROM tags. ROM tags only contain a version number, they have no revision or date fields. The recommended approach is to store a pointer to a version string in the RT_IDSTRING field of the ROM tag. This string should be in the same format as the normal version string, except without the "$VER: " prefix.

An executable file containing a ROM tag with a properly initialized RT_IDSTRING field does not need any other version information in it, such as a separate $VER-style string.

Leading Zeros. Leading zeros are not supported as part of the version and revision numbers. So "2.04" is not a valid version/revision pair as far as the version command is concerned. The reason this is so is because of the VERSION and REVISION command-line options of the Version command. These allow the version and revision of a file to be checked by a script such as:

```
Version asl.library VERSION 38 REVISION 4
```

The above command will return 0 if asl.library is greater or equal to 38.4, and 5 if it is not. Specifying:

```
Version asl.library VERSION 38 REVISION 04
```

Does the same thing. This means that from the standpoint of the Version command, 2.04 is the same as 2.4, and thus 2.04 evaluates as being greater than 2.1, which is not the desired effect. So, the version and revision numbers must not be treated as a floating-point number, but as two separate and distinct integers.

This brings up the concept of user versions and internal versions. V37 is known to the public as Release 2.04, and V39 is known as 3.0. All the version strings in the system software use version strings starting with V39. The recommended approach is as follows:

- Assign your products user-space numbers of the form 2.04 and the like.
- Assign the various components of your distribution the same version numbers as for your product. For example, "2".
- Assign each individual components of your distribution their own unique, monotonically increasing revision numbers. These numbers have nothing to do with the user-space number of your product as a whole.

An example is in order. Assume a word processor called "SliceAndDiceWord", which is at version 4.03. All the files composing the distribution would have a version number of "4", and a file-specific revision number:

```
Package number : SliceAndDiceWord 4.03
Main executable: SADW 4.1423
Support library: SADW.library 4.4129
README file    : SADW.README 4.6
```

When SliceAndDiceWord 5.0 comes out, all the version numbers of the files included with the distribution get bumped to 5, and the revision numbers start at 0 again.

Miscellaneous

This section presents miscellaneous tidbits of information worth noting when developing applications.

Overscan Names. Starting with V38, the two user-settable overscan regions are known as "Text Size" and "Graphics Size". Please use the following names in your user documentation:

```
OSCAN_TEXT     --> "Text Size"
OSCAN_STANDARD --> "Graphics Size"
OSCAN_MAXIMUM  --> "Extreme Size"
OSCAN_VIDEO    --> "Maximum Size"
```

Audio Beep. Starting with V38, the system provides standard support for audio beeps, and complete sampled sounds, as a replacement for DisplayBeep(). You may wish to rely exclusively on this feature instead of providing your own application-specific control over an equivalent feature.

ToolType Comments. We strongly encourage you to follow the convention adopted in V38 with respect to specifying all tooltypes supported by an application within the icon. Any tooltypes not in use can be commented out by putting a set of ( ) around the tooltype. This causes Workbench to ignore the tooltype, yet still leaves it there for the user to see and uncomment.

Keymap selection. If you wish to offer application-specific control over the keymap being used, you have to load the keymaps you wish to use from disk yourself. Some applications were using the SetMap command to load keymaps into memory for them. SetMap is no longer part of the system, so this technique will no longer work. The DevCon disks contain an example showing how to load keymaps from disk.

Note that if you wish to offer a list of keymaps to the user, you should be looking in the KEYMAPS: assign list for the available keymaps. To work under V37, If this assign list doesn't exist, you can look in DEVS:Keymaps. The example code shows how this can be done.

Appendix A: V39 Disk Changes

Following is a description of most of the important changes made to the disk-based system software between V38 and the initial release of V39. This doesn't cover changes made to modules discussed in other talks, such as the addition of datatypes and AmigaGuide.

C/Install

- Doing "INSTALL DF0: CHECK" on a disk formatted with dir cache now reports the disk as either "DC-OFS" or "DC-FFS" as appropriate.

C/IPrefs

- Requires Kickstart V39 and handles the new palette, pointer, and wbpattern preferences.
- Is smarter about when it needs to reset the WB screen. Now sends an initial IP_SCREENMODE message to Intuition to force the initial Workbench screen to be interleaved even if there is no screenmode.prefs file.
- Now uses utility.library 32-bit math routines and uses datatypes.library to load sound samples instead of custom 8SVX code.
- Causes much less screen flashing when changing fonts. No longer has the QUIT/S command-line option. (Removing this option allows a few K of RAM to be saved in every machine. Gives better error messages when it can't find a font or a picture.

C/Mount

- Added support for GlobVec of -2 and -3.
- No longer considers empty strings an error. This allows something like:

```
Device = ""
```

  to work. This fix will be helpful to the Envoy software. Note that this bug is in V38, so there will soon be many many Mount commands out there that don't support empty strings.

C/RequestChoice

This is a new C: program that lets AmigaDOS and ARexx scripts use the Intuition EasyRequest() feature. The template is:

```
TITLE/A,BODY/A,GADGETS/M,PUBSCREEN/K
```

TITLE
Specifies the title of the requester.

BODY
Specifies the text of the requester. Linefeeds can be embeeded using *n.

GADGETS
Specifies the labels for the different gadgets at the bottom of the requester.

PUBSCREEN
Lets you specify the name of a public screen to open the requester on.

The number of the selected gadget is printed at the console on exit, and is returned as the secondary return value (seen as the Result2 variable in the Shell).

C/RequestFile

This is a new C: program that lets AmigaDOS and ARexx scripts use the ASL file requester. The template is:

```
DRAWER,FILE/K,PATTERN/K,TITLE/K,POSITIVE/K,NEGATIVE/K,ACCEPTPATTERN/K,
REJECTPATTERN/K,SAVEMODE/S,MULTISELECT/S,DRAWERSONLY/S,NOICONS/S,
PUBSCREEN/K
```

DRAWER
Specifies the initial contents of the Drawer gadget.

FILE
Specifies the initial contents of the File gadget.

PATTERN
Standard AmigaDOS pattern. Specifies the initial contents of the Pattern gadget. Not providing this option causes the file requester not to have any Pattern gadget.

TITLE
Specifies the title of the file requester.

POSITIVE
Specifies the text to appear in the positive (left) choice in the file requester.

NEGATIVE
Specifies the text to appear in the negative (right) choice in the file requester.

ACCEPTPATTERN
Specifies a standard AmigaDOS pattern. Only files matching this pattern will be displayed in the file requester.

REJECTPATTERN
Specifies a standard AmigaDOS pattern. Files matching this pattern will not be displayed in the file requester.

SAVEMODE
Specifying this option causes the requester to operate in save mode.

MULTISELECT
Specifying this option causes the requester to allow multiple files to be selected at once.

DRAWERSONLY
Specifying this option causes the requester to not have a File gadget. This effectively turns the file requester into a directory requester.

NOICONS
Specifying this option causes the requester never to display icons (.info) files.

PUBSCREEN
Lets you provide the name of a public screen to open the file requester on.

The selected files are printed on the console at exit, enclosed in double quotes and separated with spaces. The command generates a return code of 0 if the user selected a file, or 5 if the user cancelled the requester.

C/SetFont

- Now sends ESC-C to the console handler only after doing SetFont() in the console window's rastport. The only reason SetFont ever worked was by luck, since the escape sequences were buffered by DOS and flushed only after the program exited.
- No longer crashes when passed a font name with more than 80 characters. Now returns with ERROR_BAD_NUMBER if asked for a font < 4 points. No longer opens console.device.

C/SetPatch

- Activates AA chips.
- On some systems, NMI interrupts could be caused due to hardware bugs. These NMIs can cause software/hardware to perform incorrectly and has caused major performance problems in some systems. In the V37 ROM, the NMI vector pointed at a generalized routine that saved all registers, does some work which ends up doing nothing, restores all registers, and exits. This patch will, under V37 ROMs, move that vector to point directly at an RTE.
- SetPatch no longer will read the longword after the end of each of its hunks.
- Now checks for the CPU type before trying to open the 68040.library This is so that the library does not even get loaded if you don't need it. (The 68040.library also does the check.)
- Does not turn on the caches if the intruction cache is not already on. This is such that the bootmenu option will not be "undone" by SetPatch.

C/Version

- Now does versions of gadget and datatypes classes.
- Significant change in behavior required when not using the standard version string format:

  Junk following the date parameter is printed whenever the FULL option is provided on the command line. Incorrectly formatted components of a version string are printed whenever the FULL option is provided.

  Note that as always, specifying incorrectly formatted version strings will cause the VERSION and REVISION command line options of the Version command not to work as expected. Basically, using these keywords, will result in output like "2.1 < 2.04", which is not what is usually expected.

  The incorrect version string formats are not supported by the Installer. It will behave much like the VERSION and REVISION cmd-line options of the Version command.

Classes/Gadgets/colorwheel.gadget

The colorwheel.gadget is a new BOOPSI class to create color wheels to allow color selections by the user. See colorwheel.doc for more info.

Classes/Gadgets/gradientslider.gadget

The gradientslider.gadget is a new BOOPSI class to create sliders having a gradient background. This is to be used in combination with the colorwheel.gadget. See gradientslider.doc for more info.

Devs/DataTypes

This new directory in V39 is the place where a user puts datatypes.library descriptor files.

Devs/printer.device

- Added small amount of code to support AA 8-bit HAM mode printing.
- Also now uses GetBitMapAttr() if running under V39 to obtain dimensions of source BitMap structure.

L/FastFileSystem

- Equivalent to the V39 ROM file system, including DOS\5 support. See ROM release notes for more info.

Libs/68040.library

This is a support library for 68040 systems. It addresses a number of issues with that CPU. There are no publically callable functions in this library.

Libs/amigaguide.library

The amigaguide.library is a new system module providing a means whereby developers can easily add on-line help to their applications. See amigaguide.doc for more info.

The amigaguide.library was previously available for use in commercial applications. This is the first release of the library as a part of the core operating system. Changes since the previous release include:

- All new V39-only version uses datatypes.library for object handling.
- Now has an AppWindow, plus an "Open..." menu item. To open a new document, AmigaGuide performs a LINK to it, so that you can RETRACE back through the documents that you have opened.
- Now supports word wrapping and proportional fonts.
- Now supports font attributes.
- Now supports embedded objects. This means that you can LINK to any type of object that the user has a DataTypes descriptor and class for---such as pictures and animations.
- Modified how files are located, to search for a localized help file. Now searches in the following order. For each preferred language

  ```
  PROGDIR:Help/<language>/<name>
  HELP:<language>/<name>

    PROGDIR:<name>
    <name>
  ```

  This matches how locale.library searches for catalog files. For compatibility, each directory from the AmigaGuide/Path environment variable are also searched.
- Renamed S:help.guide to HELP:<language>/Sys/amigaguide.guide
- All sub-databases are now opened localized.

Libs/asl.library

- Added support for the new WA_HelpGroupWindow Intuition tag. ASL requesters now inherit the group id of their parent window.
- Removed V37 code. The library now requires V39.
- Added support for OSCAN_VIDEO in the screen mode requester.
- Fixed rendering bug with selected items in the file requester. Depending on the size of the requester, sometimes the rendering didn't occur correctly. Fixed font requester layout bug with checkmark labels using certain fonts.
- Now monitors the Size numeric gadget more closely in the font requester.
- ASL now uses scaled checkmark imagery in the font and screenmode requesters. Also now uses ROM busy pointer instead of custom one.

Libs/commodities.library

- Fixed ParseIX() bug introduced in V38 preventing the following keys on the numeric keypad from being used in commodities: 9 ( ) / * + .

Libs/datatypes.library

datatypes.library is a new system module providing transparent data handling abilities to applications. Application developers can register their data format with datatypes.library and provide a class library for handling their data within other applications. See datatypes.doc for more information.

Libs/diskfont.library

- Marks DUPLICATE fonts using one of many free bits in the TextFont Extension. These are fonts opened by diskfont, but match some other font already on the public font list by name, Y size, relevant style, and relevant flags. The font is only marked as a duplicate if a NON-DUPLICATE occurence of this font already exists.

  DUPLICATE fonts are ignored by OpenFont() if the caller is not tag aware even if there is a font on the list which has a better implied DPI.

- Tightened up requirements for source font when scaling. Specifically, an already scaled RAM font of an exact Y size match but differing by DPI will not be used as the source for a bitmap scaled font (adding scale error, to scale error).

- The code which generates the outline fonts now does comparison for OT_DotSize when it calls OpenFont(); essentially honoring the caller's request for an OT_DotSize other than the default that diskfont will provide when creating outlined fonts. This makes it possible to have multiple outline fonts which differ only by OT_DotSize.

- Underlined outline fonts are now supported if the engine can manage it when rendering the glyphs. New tags in diskfonttag.[hi] were added for this purpose. V39 diskfont will ask for underlining if requested in the TextAttr, or TTextAttr, but if not supported by the engine, then it will simply try to open, or create a non-underlined version. So there is no change in behavior for bullet fonts in that it already opens a non-underlined version. The difference is by asking OpenFont() for a non-underlined version, it does not decide to create another copy because of the result returned from WeighTAMatch().

  Of interest, our own Text() function when used with SetSoftStyle() does a broken underline (just one pixel, but its definitely not solid), so diskfont.library V39 first asks for broken underlining from the engine, and then solid. If neither form of underlining is supported, then the above is true.

- Tag for algorithmic strike-through added to diskfonttag.[hi]; this is another feature supported in Final Copy, and could be requested if wrapped in a bullet interface.

- Underlining off, double solid, and double broken underlining also defined for the new OT_UnderLined tag.

  Underlining is not an intrinsic style for outline fonts. This does not actually absolutely have to be the case, but we lack a tag to indicate intrinsic underlined style now, so this is not a new behavior, or limitation. The assumption being that underlining is one of those things that is probably best done algorithmically by the application, or for the purposes of diskfont, by the engine. Therefore this also leaves open the possibility of placing the underlining code within diskfont for engines that do not support it should we wish to do so in the future.

Libs/iffparse.library

- Fixed three bugs in the OpenClipboard() function. First bug was that if the clipboard.device couldn't open, two calls to FreeSignal() were made with uninitialized values as parameters. The result of this was a corrupt signal mask in the Task field. Second bug what the OpenDevice() was called with an IO request that didn't have a valid MsgPort pointer in it. Finally, the two message ports allocated by the function were not being initialized correctly and would cause a system crash if a message ever got to either of them.

Prefs/#?

- Now restore task->tc_UserData on exit of prefs editor.

Prefs/Font

- Default for WB icon text mode when there is no .prefs file is now JAM2 instead of JAM1. This matches the ROM.

Prefs/IControl

- Added Mode Promotion gadget when running on AA machines

Prefs/Input

- "Show Double-Click" now operates asynchronously. That is, while the double-click sample is displayed, it is still possible to click on other gadgets.

Prefs/Overscan

- Now bounds check values read from prefs file against what the gfx database says. This avoids problems with V38 overscan prefs files, and with running VGAOnly or not.

Prefs/Palette

- Brand new interface using the colorwheel and gradientslider gadget classes. Also allows pen spec editing.

Prefs/Pointer

- Now opens on the Workbench screen if it can obtain three exclusive pens.

- Allows editing of the Normal and Busy pointers and supports Low-Res and High-Res pointer resolutions.

- Uses datatypes.library for clipboard support.

- Semaphore around save during Test. This is to prevent confusing error requesters if the user presses the Test button several times while waiting for the pointer to appear.

- Added "Load Image..." menu item in the "Edit" menu.

- Uses (SA_LikeWorkbench, TRUE) when using a custom screen.

- Increased width of window by 20 pixels to make room for longer strings.

- Added NOREMAP tooltype and command-line option. This causes paste and import image to not remap.

Prefs/ScreenMode

- Now skips the default monitor when scanning the mode list instead of skipping PAL or NTSC. This means that the prefs file will never contain "default monitor" as a mode, it will always contain an explicit mode.

Prefs/Sound

- Now uses datatypes.library for sound loading and playing. Also uses a DataTypes filter in the load sample requester so that only sounds will be shown.

Prefs/WBPattern

- o Added support for picture backdrops.
- o Added support for Workbench screen backdrop.
- o Defined a more efficient chunk for the preference data.
- o Fully supports the First4/Last4 Workbench color scheme.
- o Uses datatypes.library for clipboard support.
- o All internal bitmaps now max out at a depth of 3 (less memory usage).
- o Rearranged the gadgets.
- o Semaphore around save during Test. This is to prevent confusing error requesters if the user presses the Test button several times while waiting for the pattern to appear on the Workbench.
- o Added "Load Image..." menu item in the "Edit" menu.
- o Added NOREMAP tooltype and command-line option. This causes paste and import image to not remap.

S/Startup-Sequence

- o Removed "Echo" statement. That means the initial Shell window will no longer be displayed on bootup on a standard system. This means a noticeable decrease in bootup time on slower machines (it takes about a second to open and close the shell window). It means even more savings in time when the WB has more bitplanes. Finally, it also looks a heck of a lot more professional.
- o Added C: in front of every pertinent command. This speeds up the S-S noticeably on floppy systems, since normally the current directory is scanned first for a command, and then C:. By prefixing C:, it no longer checks the current directory making things a whole lot faster.
- o Added deferred assign for HELP:.
- o Added multi-assign of LIBS: to SYS:Classes in support of the new disk-based BOOPSI classes.
- o Now runs VGAOnly monitor file if it is installed.

Storage/Monitors/#?

- o Added DblNTSC and DblPAL in support of AA scan doubling.
- o Added VGAOnly monitor.
- o Added AA support throughout.

System/DiskCopy

- o Enabled code to correctly report read errors. Under V38, read errors are reported as write-errors.

System/Format

- o Requires V39.
- o Added directory caching support in the form of an extra gadget in the main Format window, and some new command-line options. Command-line template is now:

  DEVICE=DRIVE/K/A,NAME/K/A,OFS/S,FFS/S,INTL=INTERNATIONAL/S,
  NOINTL=NOINTERNATIONAL/S,DIRCACHE/S,NODIRCACHE/S,NOICONS/S,QUICK/S

- o Uses GTLV_MakeVisible tag in its device listview.
- o Fixed bug where long volume names would run off the right edge of the main window by using the new GTTX_Clipped gadtools tag.

System/Intellifont

- o Renamed from Fountain to Intellifont.
- o Bare minimum needed to fix visual bug evident when running under V39 gadtools; no longer uses GTLV_Selected tag, though still does not use V39 style selection via GTLV_ShowSelected tag.

Tools/Commodities/MouseBlanker

This is a mouse blanking commodity. It blanks the mouse when you start typing and unblanks it when the mouse is moved or any of the mouse buttons are hit.

Tools/HDToolBox

- o Brand new GadTools interface.
- o Now determines the version number of a file system automatically instead of requiring it to be entered manually.
- o Fixed a great number of bugs.
- o When CD_ROM drive was hooked on the SCSI bus and the disk was inserted, HDToolBox hanged up beacause the program assumed only 512 bytes blocks. It Now checks the size with the "READ_CAPACITY" command, and if it fails, sends "INQUIRY" command and gets device type. If it's a CD-ROM device, sets block size to 2048.

Tools/IconEdit

- o Implements the First4/Last4 Workbench color scheme.
- o Uses datatypes.library for clipboard support and picture loading/saving. Therefore, you can now load any picture type that you have a class for. When loading or pasting a picture, the colors are mapped to the Workbench GUI colors (either 2, 4 or 8 colors). 8 color icons will work with any depth Workbench screen.
- o Default clipboard viewer is now "Multiview CLIPBOARD".
- o Checks to see if icon already exists when saving a picture or source file and won't overwrite it.
- o File requester's initial position now matches the preference editors.
- o Now draws the coordinates in the window title bar using the pen spec.
- o Was reloading the icon image after save, even if save failed.
- o Will now properly undo after loading a new icon image.
- o Added NOREMAP tooltype. This causes paste and import image to not remap.
- o Fixes Enforcer hit caused when dropping Drawer icon (without a drawer) into the AppWindow.

Tools/ShowConfig

- o Added AA chipset support.

Utilities/MultiView

This is a generic view-anything utility that uses datatypes.library for its object handling.

3.0 Installation Procedure

- o It is no longer necessary to boot from the Install disk.
- o Install Languages no longer exists. This is now merged into the main installation script.
- o Install Printers no longer exists. The user can easily drag a printer driver icon from the Storage disk to DEVS:Printers
- o The disk now includes the 68040.library, which gets copied to the HD when performing an install
- o The various Prod_prep scripts were enhanced and work better.
- o There are many little utilities in the C: directory which aid the main install script to do a better job.

Changes to the install script include:

- o Requires V39 ROMs
- o Now integrates the old "Install Languages" script. When starting the main install script, you are now asked whether to perform a complete installation, or only update the languages.
- o It now virtually always makes the right decision as to where the WB files should be copied. As a result, if the user picks Novice mode, he will not be asked where the files should go.
- o Now preserves the tooltypes of many icons. This will avoid zapping the stuff put there by users, making the update process much more transparent.
- o Now preserves the locations of most files. For example, if the user moved Blanker in WBStartup, it will be correctly updated in WBStartup, and left there. In V38, the copy in WBStartup would be deleted and the new version installed in Tools/Commodities.
- o Now preserves left out icons. The UpdateWBFiles program ensures that the contents of the .backdrop file are up to date which removes the need to delete the file.
- o Now updates V37 font prefs files to V38/V39 format.
- o Now deals with the "Storage3.0" disk
- o Now asks which keymaps to install.
- o Installs the 68040.library.
- o Knows about systems with MapROM, and installs a special startup-sequence.
- o On an NTSC system, always copies the NTSC monitor to DEVS:Monitors, and copies PAL to DEVS:Monitors on PAL systems.
- o When installing languages, deletes languages the user did not choose. This is to remove old V38 files that might be hanging around.
- o Does better sniffing to determine whether an A2090 is present, and displays extra info to inform the owner that some files need to be copied.
- o Will optionally reboot the system for the user at the end of the installation procedure (after asking permission first, of course).
- o Asks all its questions up front. Once the questions are answered, it starts the actual installation.

- o Now deletes all C= printer drivers and keymaps before starting the installation process. This was previously done right before re-installing them. This is to increase the likelihood that things will fit on the HD by deleting obsolete stuff ahead of time.
- o Does slightly better icon positioning to avoid things moving "by themselves".
- o Now copies the A2232 port-handler and A2232 aux-handler when needed.

Changes to the HDSetup script include:

- o Will now partition the HD and format it automatically instead of requiring a reboot.
- o The partitions created are now called Workbench (HD0) and Work (HD1). HD0: is created as 8M.
- o DOS\3 is now used by default, and the DMA mask is set to 0xfffffffe instead of 0xfffffffc
- o The version number for the file system being installed on a machine is now extracted from the file system load file instead of being hardcoded in the prep scripts
- o After completing the reselection on, reselection off, or update superkickstart option, the system is rebooted automatically (after asking the user of course).
- o Updating an A3000 super kickstart no longer requires the user to hit RETURN. Errors are also detected now.

Appendix B: Release 3.01 Disk Changes

Following is a description of most of the important changes made to the disk-based system software between Release 3 and Release 3.01. This doesn't cover changes made to modules not discussed in this talk. It also doesn't cover any changes after December 1992. Since 3.01 is still under development as of this writing, more changes are likely to be made to the software prior to release.

C/ConClip

- Now consumes over 3100 bytes less RAM when running. That means over 3K more RAM in every system we ship.
- Localized
- Now keeps iffparse.library and clipboard.device closed whenever they are not in use.
- Added CLIPUNIT synonym to existing UNIT command-line option. This is to make it comply with the style guide and be more consistent with other programs such as WBPattern. The template is now:

```
CLIPUNIT=UNIT/N,OFF/S
```

C/Copy

- Now correctly handles failure of SetProtection(), SetComment(), and SetFileDate(). Because SetComment() fails on NFS and on CrossDOS, SetFileDate() never was done on files, making the CLONE option not fully work. All three functions are considered failures only if IoErr() reports something different than ERROR_ACTION_NOT_KNOWN.
- In case of error after a call to DupLock(), an error code was being set always to ERROR_NO_FREE_STORE instead of using the result of IoErr().
- The default size for the buffers used was always equivalent to (BUF=0) which caused the buffers to be the size of the files being copied. This was contrary to the docs, and caused problems when copying large files through Envoy, as it could easily eat up all the memory in the system, not leaving enough for the memory needed by Envoy. The default size is now BUF=128 which gives a 64K buffer.

C/IPrefs

- Added needed support for the new display position control now offered by Overscan prefs.
- If there is no icontrol.prefs file, IPrefs sets the default for Mode Promotion to ON. That means machines will have mode promotion on by default.
- Now ensures that requested width, height, and depth for Workbench, fall within the allowed range as defined in the graphics database.

C/Mount

- Only change is the version number. Versions 39.1 and 39.2 of this program were accidentally numbered 38.1 and 38.2.

C/Protect

- Fixed error reporting. When used with wildcards, it would generate errors of the form "Can't set protection for #?" instead of giving a descriptive file name, and cause of error.

C/SetPatch

- Due to a bug fix late in the game, the filesystem broke with ExAll() on DCFS file systems if the lock passed in was not Examine()ed first. This patch fixes this by forcing an Examine() on a lock before the ExAll() is called. It only does this on systems with V39.22 dos.library since only that ROM has the problem with DCFS ExAll().
- ChangeVPBitMap() was not properly doing the bitplane swizzle needed for 8-bit HAM mode. The reason that this didn't show up earlier is that if the relationship between the bitplane pointers was the same in the original and new bitmaps, the bug would correct itself. So, you would only see this bug if your memory was fragged, or if some other allocations got in between the separate allocations of the bitplanes.
- ScrollVPort() had the same bug as ChangeVPBitMap().

  The patch locks the ActiViewCprSemaphore, swizzles the bitmap, calls the old entry point, and unswizzles the bitmap, when called on a HAM-8 viewport.
- The patches for graphics.library/ScrollVPort() and ChangeVPBitMap() now bump the graphics revision number to 90 if it is 89. This is so that installing the setpatch on an A1200 or A4000 will not break people using work-arounds for the bug.
- When calling BltBitMap() with both source and destination interleaved, and a mask of -1, the low byte of d7 would be changed to the bitmap depth on exit. Patch saves d7, calls old, and restores it.
- Added the graphics monitor/view association hash patch.
- BltMaskBitMapRastPort() used to interpret the mask data incorrectly if both the source and destination bitmaps were interleaved.
- Now includes the CP2024 Conner patch from the V37 SetPatch.

C/Version

- Obtaining the version of printer.device while it is loaded in memory would trash memory. This is because printer.device doesn't initialize its lib_IdString. Although Version was checking for NULL, the lack of string was causing problems in the output routines.
- Fixed the FILE option which was causing the version numbers not to be displayed.

Classes/Gadget/colorwheel.gadget

- Querying the wheel for an explicit red/green/blue value would not get the most up to date brightness value from the gradient slider causing things to get slightly out of sync.
- Fixed bug where trying to open the class library under 1.3/2.0 would cause a system crash instead of simply failing.

Classes/Gadget/gradientslider.gadget

- Fixed bug where trying to open the class library under 1.3/2.0 would cause a system crash instead of simply failing.

Devs/mfm.device

- Fixed small problem of the wrong return results when opening a device with an invalid unit number.
- Changed the format function to update the physical track and invalidate the in-memory buffer,

L/CrossDOSFileSystem

- Booting off a CrossDOSFileSystem formatted disk does not crash an IBM machine (including CrossPC). It now presents a message "CrossDOS non-bootable disk!".
- Now supports formatting a compatible MS-DOS hard disk partition. It is still not bootable.
- Now prevents an MS-DOS hard disk partition to be formatted if not properly partitioned first.
- Fixed problem with method of reporting results when asked to read or seek past end or beginning of file. Now follows V37/V39 FFS method.
- Fixed problem with parsing a file or directory name with NULLs instead of the usual SPACEs for blanks. This occurs in PC-DOS not in MS-DOS.
- Fixed bug that translated extended ASCII characters incorrectly when either the INTL or DANSK translation tables were selected.
- Added stack size checking.
- Fixed a bug that appeared when using the 'assign' command with this FS. A DOS deadlock could occur in certain circumstances. Added a delayed volume node addition and deletion algorithm.
- Fixed a small problem of freeing memory allocated by the process spawned by the FileSystem_status process. This could cause memory shared to be reused when it was still needed. The fix was to add the memory entry list to the filesystem_status process which is the last process to exit. This only occurs if the target device could not be opened.
- Fixed a problem that never showed up yet. Internally reproduced only.

L/port-handler

The main reason for this release is to fix problems that PRT: was having with Envoy printing.

- Now has a version string.
- Now requires V37 ROMs.
- Now supports ACTION_IS_FILESYSTEM and ACTION_FINDUPDATE.
- Once loaded, it remains in memory permanently which fixes many problems. For example, if any of SER: PAR: or PRT: had been active, double-clicking on Format, DiskSalv, or CrossDOS would cause Enforcer hits, or crashes on a machine not running Enforcer. This also fixes "copy foo to PRT:" when PRT: is sending data to the envoyprint.device.
- When first loaded in memory, the handler patches its seglist pointer into the DOS device node of other related handlers. That is, if the handler is started as PRT:, it patches its seglist into the DOS nodes for SER: and PAR:. This means that all these devices end up sharing the same seglist for the port-handler, which can save memory and disk-loading time. It patches its seglist in any device node having "L:port-handler" as handler name.
- This port-handler replaces both the standard port-handler on the Workbench, and the A2232 port-handler. As such, the handler supports being activated through a mountlist entry, and will process the BAUD, CONTROL, DEVICE, UNIT, and FLAGS keyword from a mountlist.
- Because this port-handler replaces the A2232 one, it is possible to specify the baud rate and control flags when opening a serial unit. For example: Open("SER:9600/8N1",...);

L/FileSystem_Trans/#?

- Modified a few values for the INTL and DANSK translation tables to default to the SPACE character for untranslatable characters instead of NULLs.
- Added a Mac translation table. This lets an Amiga read Mac ASCII files. This translation type can be selected from the CrossDOS commodity.

Libs/asl.library

- In screen mode requester, fixed formatting string so that horizontal scan rates with a decimal value less than .1 now come out right. That is, 29.02 was coming out as 29.2.
- Fixed bug where trying to open the library under 1.3 would cause a system crash instead of simply failing.
- Selecting Restore in the screen mode requester now correctly resets the "Overscan Type" and "AutoScroll" gadgets.
- Fixed bug where the file requester's drive LED would remain lit after creating a new directory when in save mode.

Libs/commodities.library

- Fixed bug present since V36. Input events of type IECLASS_NEWPOINTERPOS did not get their extended data properly copied in various places, including through AddIEvents(). This resulted in the copy of these input event containing pointers possibly pointing to garbage.
- Fixed bug where NEWPOINTERPOS was not considered a valid class by ParseIX().
- Input events inserted into the input food chain as a result of the action of a translator object are now marked with the same time stamp as the input event which activated the translator object.
- DisposeCxMsg() was accessing a global list without proper locking. This was quite nasty, since the list is typically accessed from both the commodity input handler, and from commodity programs running asynchronously.
- Now nulls out the ie_NextEvent field of input events it copies. All input events generated by the library are copies of other events, and the ie_NextEvent value was left set to the original event's value. That means commodity was spewing out a lot of input events with bogus ie_NextEvent fields. This could cause crashes and confusion.
- InvertKeyMap() now sets the ie_SubClass and ie_TimeStamp fields to 0 in the input events it generates. These were being untouched which means they could contain garbage.

Libs/iffparse.library

- Fixed bug where trying to open the library under 1.3 would cause a system crash instead of simply failing.

Libs/68040.library

- During the setup of the MMU tables, if an expansion card was 0 bytes in size it would cause the MMU table setup to fail. 68040.library would continue to work but the magic MMU setup that would have been needed for the Zorro-III boards would not be done. This has been fixed.

Monitors/#?

- Now use a new algorithmic approach to generate the database entries. The result of this is that the amount of disk space taken up by these new monitors is 140 blocks, compared to 176 blocks they used to take.

More important is that the dimensions of Multiscan, Euro72, Super72, DblNTSC and DblPAL are now greater than they used to be. This is especially important for DblNTSC and DblPAL, which are meant to resemble NTSC and PAL as closely as possible. The Dbl.... monitors are now 720 pixels wide for MaxOScan, compared to 724 for NTSC/PAL, and the 676 they used to be!

Also, if not running with VGAOnly, the "dead" part of the display on the left hand side of the screen is now useable as VideoOScan.

- o Defined ScanDoubled versions of the Multiscan, Euro72 and Super72 monitors for better coercion on AA machines.
- o All these monitors should now work under ECS, and give even greater dimensions.
- o No AA modes should be in RAM on non-AA machines.
- o Changed the names of the monitors from DoubleNTSC/PAL.monitor to DblNTSC/PAL.monitor to be consistent with the ModeID name strings.
- o All the monitors now work only with graphics.library V39.
- o With Kickstart 39.106 and VGAOnly, DblNTSC/PAL screens can only be 704 pixels wide max, whilst under the current kickstart they are 720 pixels wide. The DblNTSC/PAL monitor now checks for this.
- o Cleared up an enforcer hit with adding the A2024 after iprefs was run, and promotion was enabled.
- o Sprites were not being clipped properly in PAL A2024 modes when they crossed the panel boundaries. This was causing screen trashing when the pointer was at certain positions because the pointer would end up in the A2024's special control line.

Prefs/IControl

- o Removed "Preserve Colors" gadget. The "Avoid Flicker" gadget has been moved into the "Miscellaneous" gadget group instead of being by itself in the "Coercion" group.
- o If there is no prefs file, mode promotion is now turned on by default.

Prefs/Input

- o Now uses GTLV_MakeVisible tag to ensure the currently selected keymap is visible in the listview.

Prefs/Overscan

- o On the edit screen, changed the label of the left gadget from "Use" to "OK".
- o Added code to support the new display positioning control. When you enter the overscan editing screen, if the mode supports it, there are four arrows that cause the entire visible portion of the display to move around. This is a wonderful new feature for DblNTSC, allowing sync-dependant centering of display modes. It is also possible to move the display by using the cursor keys.
- o Changed the background color on the edit screen from black to dark grey. This makes it much easier to detect when the display is being pushed too far off the right edge of the display, as the color of the display turns quite dim.
- o Made the edit screen SA_Exclusive in order to avoid folks dragging it down and revealing their WB. Since the program now makes dynamic changes to hsync and vsync, the WB can look bad when the edit screen is pulled down.
- o Made the edit screen SA_Interleaved.

Prefs/Palette

- o When incapable of displaying a colored color wheel, the program now puts up a gadget labelled "Color Wheel..." in place of the actual color wheel. Clicking on this gadget causes the color wheel to come scrolling up from the bottom of the screen until it fills the whole display. This color wheel screen lets the user edit the current color using the wheel. There is a pair of "OK/Cancel" gadgets to accept or reject the new color selection. Clicking on either gadgets causes the color wheel to scroll off the bottom of the display. Kinda neat :-)
- o The sample section of the program now shows a sample screen title bar.
- o When opening on a custom screen, no longer has a window border and title bar.
- o When switching modes from 4 to multicolor and back, will no longer present an empty window if it is not capable of creating its new set of gadgets. The program will exit instead.
- o Selecting a color > 4 in the main palette, and switching to 4 Color Settings would not correctly redraw the gradient slider to use a color within the available selection.
- o Now uses color conversion routines from colorwheel.gadget instead of having inline duplicates.
- o Fixed incorrect flags set in some menu items so that click select of these items worked incorrectly.
- o Was letting you select between 4 and Multicolor Settings, even when running on a 4 color screen incapable of displaying the multicolor settings.
- o When running on an A2024 display, it no longer displays a color wheel, nor lets you display one.
- o Clicking in the sample section of the program causes the pen associated with the item clicked on to become selected in the pen list. That is, clicking on the sample window title bar will automatically select the "Active Window Title Bars" pen within the pen listview.

Prefs/Printer

- o Now uses GTLV_MakeVisible tag to ensure the currently selected printer is visible in the listview.

Prefs/PrinterPS

- o The default margins for graphics printing are now 1 inch on the left and right, instead of 1 inch on the left and 2 on the right (this was due to a typo).
- o When in the Graphics Scaling page, selecting "Last Saved" from the menu would not cause the sample graphics to be redisplayed with the new values.

Prefs/ScreenMode

- o Fixed formatting string so that horizontal scan rates with a decimal value less than .1 now come out right. That is, 29.02 was coming out as 29.2.

System/Format

- o From within the device selection listview, it was possible to double-click on two different devices, and have Format accept the selection. A check is now done so that both clicks of a double-click occur on the same device.
- o When formatting from Workbench, Format now ignores any trailing colons in the volume name. It would get all confused if someone tried to name a disk "Hello:" instead of simply "Hello". This fixes the most common error automatically.

Tools/Calculator

- Now gets the window title string from the current catalog instead of having it hardcoded to "Calculator".

Tools/Commoditites/ClickToFront

- Now also filters mouse clicks that come through as IECLASS_POINTERPOS and IECLASS_NEWPOINTERPOS, in addition to IECLASS_RAWMOUSE. This fixes the bug where ClickToFront did not work with events generated by tablet drivers.

Tools/Commodities/CrossDOS

- Fixed bug where the window would not open when there was no L:FileSystem_Trans directory, or no file within it.
- Fixed incorrect cleanup path in case of errors while scanning directories. The bug would cause a crash the next time the window was opened.
- Now only notifies CrossDOS handler tasks of changes in settings when new settings differ from the old ones.
- No longer has the CrossDOS semaphore locked when it locks the DOS device list. This could potentially cause deadlocks.
- Fixed bug where translation files were opened and read, but never closed.
- Fixed bug that would cause a crash under V37 or harmless Enforcer hits under V39. To reproduce, start the CrossDOS commodity and copy a large file to PC0 (devs:Kickstart for example). While the copy is in progress, pick PC0 from the CrossDOS commodity and change its text filtering setting. Now click the close gadget of the commodity program. The window stays open until the large write operation completes. As soon as this happens, the window closes, and a crash occurs under 2.0.

Tools/Commodities/FKey

- When there was a key sequence in FKey with a command such as "Insert Text" with a string associated with it, switching the command to something else, and coming back to "Insert Text" would cause the current string to be "forgotten". To get it back, you had to click in the text gadget and press RETURN. This is now fixed.
- Selecting a command such as "Insert Text", then typing in a string as argument would cause that string to be bound to the key sequence forever. That is, even if the command was switched to something like "Cycle Windows", the string argument would be saved out to disk and reloaded in memory the next time the program ran. This was incorrect as the command didn't have anything to do with the string. String arguments are now discarded for those commands that don't support them.
- The active key sequence is now correctly highlighted in the listview.

Tools/HDToolBox

- Now supports Host ID in "Partitioning" screen. Now multiple machines can share a single disk using different partitions.
- Added CHECKBOX_KIND gadget for allowing a block size other than 512 bytes in "File System Maintenance". To change file system block size in "File System Characteristics", select this gadget first.
- Changed from BUTTON_KIND gadget to CYCLE_KIND gadget for "File System Type". They get labelled with their dostype as found in the file system resource list.
- Added CHECKBOX_KIND gadgets for "Fast File System", "International Mode" and "Directory Cache".
- Added CYCLE_KIND gadget for file system block size.
- Recognizes the "Help" key now.

HDSetup/HDSetup

- Localized the name of the "Work" partition following request from Germany.

Install/Install

- No longer copies the A2232 port-handler from the install disk when an A2232 card is detected. The 39.1 port-handler handles the A2232 automatically

Appendix C: V39 ROM Changes

Following is a description of most of the important changes made to the ROM-based system software between V37 and the initial release of V39. This doesn't cover changes made to modules discussed in other talks, such as graphics and Intuition.

BootMenu

- Brand new interface featuring 4 different displays:

  Main Page: Lets you select between "Boot Options...", "Display Options..." and "Expansion Board Diagnostic...". Clicking any one of these brings up the corresponding page. The "Boot" gadget resumes the boot operation using the options selected in the other pages, and the "Boot With No Startup-Sequence" gadget does the same, except it doesn't execute the startup-sequence.

  Boot Page: Lets you control boot-related options. The listview on the left lets you pick which device to boot from. The one on the right lets you enable/disable devices in the system. There is also a "Disable CPU Caches" gadget. It turns off the CPU caches for the current boot, which saves a lot of games that break on 68040 processors because of the big caches. Use or Cancel bring you back to the main page.

  Gfx Page: Lets you pick what mode to boot in, and what chip set to emulate. Use or Cancel bring you back to the main page.

  Diag Page: Lists all the boards in the system and shows their state (Working or Defective). This page is automatically brought up during system bootup if a board is defective.

- Hitting any key toggles the display between NTSC and PAL. A message on the main page indicates this fact.

cia.resource

- Changed the priority of the interrupt servers to +120 such that they don't miss interrupts.

con-handler

- Fixed a low-memory trashing problem were CON: would signal a NULL task.
- It no longer will use proportional fonts (or rather fail at trying to use them) when opened on a public screen.
- It is no longer possible to size the window such that the entire interior goes away.
- Fixes the title bars (the parsing routine wasn't skipping over delimiters). This also fixes the "funny" results you would get with malformed CON: lines (like con:0a/0/640/200/title having a title of "200").
- Fixes negative sizes in the window spec. Height < 0 gives you max displayable height.
- Always opens window big enough for one line of text.

console.device

- CMD_CLEAR fixed, was broken in V37.
- Uses screen dimensions (rounded off to nearest byte width) to calculate maximum character map width instead of using bm_BytesPerRow. Likewise uses screen height instead of bm_Rows.
- First pass at the scrolling optimizations. Recalcs scroll mask at reset time (set to defaults), at full clear screen time (FF, or HOME/CLS), and whenever you set new pen/cell/background colors via the standard SGR sequences.
- Rework how conceal mode works - no longer sets rp_Mask to 0 which also disabled ALL output, including scrolling, clearing, etc... not good for character mapped consoles in particular since it causes the display & map to lose sync.
- Scroll DOWN no longer tries to fill in vacated portions of the window with text in the off-screen buffer (if any). This was useless as a scrollback feature, could crash if you scrolled down more than one window's worth of text (ouch), didn't perform a window refresh (thereby leaving hidden characters in the visible map until you resized, or revealed), and was inconsistent with SMART REFRESH consoles - the application using scroll down would reasonably expect that the vacated portion is entirely empty.
- First pass at breaking up large CMD_WRITEs. Now unlocks layers approx every 256 characters (this is simple, low-overhead code which does not try to be exact). Recalcs everything when layers are relocked.

  Helps quite a bit. No more locks for the entire file when doing COPY foo *. Makes console much more friendly; you can now resize windows (only minor delay) during long writes, click in other windows, etc.

  Note this simple code won't work for a long stream of text with no control characters; this however is extremely rare. Even text files have LF's, however in any case the problem is no worse than it was before.
- The above feature makes it possible to cheaply break a CMD_WRITE, hence making it possible to easily fix the DisplayBeep() deadlock bug. DisplayBeep() is now postponed a few CPU instructions until after layers are unlocked.
- First pass at modifying mouse tracking code (drag selection) to use input events instead of PeekQualifier().

  This code change was added so tablet drivers, and commodities can be used in character mapped consoles with selection capability. PeekQualifier() returns only what input.device thinks the qualifiers are; not what's seen by applications which watch/use the input stream.
- Now supports a new private sequence ("ESC[s") which sets the current SGR settings as your defaults. This affects ESC[0m (reset all SGR settings), ESC[39m (reset default primary pen), and ESC[49m (reset default secondary pen [cell color]).

  Text style info, and reverse mode (on/off) are also saved, and hence restored when ESC[0m is sent.

  This is intended as a user sequence for use in your shell-startup; it allows you to use other colors/settings, and not have these constantly reset by programs like MORE, LS, etc. I'm recommending it not be used by applications; only users for their shells. An application which can deal with this problem of SGR settings should continue to do whatever it is doing now. ESCc (reset console) does however reset the default SGR settings to their true defaults.
- Downcoded pack.c. Is many times faster (if the maps are not disorganized; the maps become disorganized as text is scrolled off screen, so in these cases an initial resize can still take a moment - didn't want to touch that code though). For an organized map, resizing even very large windows (e.g., Moniterm size) with 8x8 or smaller fonts (so we have a really large map) is virtually instanteous on a 3000, and nearly so even on 68000 machines. It still takes time to redraw

the text (limited primarly by the Text() function), but the time needed to pack, and unpack the map is a fraction of what it was.

- Borderfill code added so ESC[>#m fills to borders if an explicit line length and/or page length have not been set. No change to cu_XRExtant or cu_YRExtant in public portion of console unit structure.

  Border refers to the area outside of the normal console rendering area up to the window right/bottom borders. The size of this area is 0-N pixels where N is a maximum of the font width-1 or height-1.

- OpenDevice() now fails if trying to open a character mapped console, but memory can not be allocated for the map. In V37, OpenDevice() returned success for this case which left you with a half functioning console window - clearly confusing for the user, and virtually worthless because of the lack of refresh info needed to fix up damage.

  OpenDevice() still works the same if you have a SIMPLE_REFRESH window, but did not ask for a character map (uncommon, but doable).

- Also fixed a bug which you may never see now that the above code was added; clearing a simple refresh window which lacked a character map cleared a garbage rectangle; layers prevents this from being a crash, and code elsewhere inhibited negative rectangles. The bug exists in 1.3 also, and was partially fixed for 2.0; the bug use to be apparent in SUPER_BITMAP windows, and because the case of SIMPLE_REFRESH without a map is rare for console.device, the bug has probably never been noticed (found during memoration testing).

- Removed code which checks to see if the application had drawn over the cursor in a console.device opened by the application. The kludge did a ReadPixel() of every pixel where the cursor was drawn, and if any bits did not match the expected color (also modified by pattern), cursor drawing was turned off for that console for as long as that console window was opened.

  Applications (few) which draw over the console cursor, but do not explicitly turn it off will now have a patterned rectfill the size of the cursor (generally 8x8) in the upper left hand corner of the window if the window is deactivated. This is a minor visual problem, though not one which should cause anyone to crash, or not run. The problem will also never be noticed if the console window is not deactivated.

dos.library

- Fixes a bug in Open() where if the path was more than 127 characters long, a random byte of memory would be trashed.

- Has support for fib_OwnerXXX for the networking people. ExAll() supports ED_OWNER.

- Added ExAllEnd().

- Added SetOwner().

- Fixed overrun error in FGets() (if no newline or EOF, it reads one byte too many into your buffer - workaround for V36/37 - allocate buffer 1 byte larger than passed in.

- HUNK_RELOC32SHORT now works at the right value (1020). Also added a 32-bit PC-relative reference mode, mainly for >= '020-only executables.

- Added GVF_SAVE_VAR. For SaveVar(), it will now do the same actions for ENVARC:whatever, as well as ENV:.

- Fixed character classes in ParsePatternNoCase(). The classes weren't being promoted to upper case (i.e. [a-z] should have become [A-Z]). Note that only ParsePatternNoCase() was affected by this bug, not MatchFirst()/MatchNext()

- Fixed FreeDosObject(xxx,DOS_CLI). It wasn't freeing the strings associated with the CLI.

- FindCliProc(0) now returns NULL, even though it's an invalid CLI number.

- For people passing in an RDArgs structure but no RDA_Buffer to ReadArgs(), it now properly clears out RDA_Buffer on FreeArgs(), so you can reuse it safely (without having to clear it out yourself).

- StrToLong() was returning the number of white-space characters if no digits were found. It now properly returns -1.

- Fixed GVF_DONT_NULL_TERM for global variables.

- The initial console window on bootup now opens the size of the Workbench DClip.

- Fixes the CliInitNewCLI() open of S:Shell-Startup when no FROM file is specified.

- Fixed "Copy CONSOLE: foo" by making GetDeviceProc() know about CONSOLE:.

- Localized "Software Failure".

filesystem

- Added support for DOS\4 and DOS\5 file systems which offer directory caching. DOS\4 and DOS\5 are orders of magnitude faster at directories than other file systems, since they keep a cached copy of all the ExNext()/ExAll() data appended to the directory. This does require a few extra block accesses on create and delete, and also after modifying the file (in Close()). Create speed dropped about 30%. However, directory speed is 7-20 times faster.

  For floppies, this usually means that dir or list take about 3/4-1 second to start, and then you get most or all of the directory instantly, or within 1/2 second or so (it may take 1 or 2 seconds for really big directories).

- Fixed a bug with delete for non-DOS\5 partitions.

- Fixed a random memory trash in the filesystem in a race condition. When two renames hit just the right timing, one has to wait on the other, and the wait routine used the wrong (garbage) register to get the head of the list. This trashed 1 longword of semi-random memory, and then hung the rename forever.

gadtools.library

- LayoutMenus() and LayoutMenuItems() recognize some new tags to support NewLook menus:

  GTMN_NewLookMenus (BOOL): requests NewLook menu treatment.
  GTMN_Checkmark (struct Image *): checkmark you'll use in menus
  GTMN_AmigaKey (struct Image *): Amiga-key image you'll use in menus

  Basically, if you open your window with WA_NewLookMenus, also lay out your menus with GTMN_NewLookMenus. If the menu-item font will be the screen's font, that's all you need to do. If the menu-item font is something else, you must create a checkmark and an Amiga-key image, and pass each one to both Intuition (WA_Checkmark and WA_AmigaKey) and to GadTools (GTMN_Checkmark and GTMN_AmigaKey).

  GTMN_FrontPen is now recognized. If GTMN_NewLookMenus is specified, this attribute defaults to the screen's BARDETAILPEN, else it defaults to "do nothing", which allows the GTMN_FrontPen tag that may have been passed to CreateMenus() to still hold.

- STRING_KIND, INTEGER_KIND, and BUTTON_KIND gadgets now support the GA_Immediate tag.

- You can now put an arbitrary command string in the right-hand side of a menu, where the Amiga-key equivalent goes. To do this, point the NewMenu nm_CommKey field at the string (eg. "Shift-Alt-F1), and set the new NM_COMMANDSTRING flag in nm_Flags.

- If a window has multiple checkboxes or radio buttons, a separate image is no longer allocated for each one.

- The bevel box of the slider and listview now refresh with the gadget, instead of with GT_RefreshWindow().

- Scrollers with GA_RelVerify set weren't sending IDCMP_GADGETUP messages when the arrow buttons were released.

- GadTools now uses SetABPenDrMd() when advantageous.

- New GTMX_Scaled and GTCB_Scaled tags instruct GadTools to scale the mx button and checkmark respectively to the dimensions supplied in the NewGadget ng_Width and ng_Height fields. Under V37, or in the absence of these tags, the dimensions are fixed. Added #defines for those dimensions.

- GadTools now has a GT_GetGadgetAttrsA() function (and a GT_GetGadgetAttrs() varargs version). This function retrieves attributes of the specified gadget, according to the attributes chosen in the tag list. For each entry in the tag list, ti_Tag identifies the attribute, and ti_Data is a pointer to the long variable where you wish the result to be stored.

- Checkboxes now return their "selected" state in the IntuiMessage->Code field.

- Many new tags for CreateGadgetA() were added:

  GTTX_FrontPen and GTTX_BackPen to let the color of TEXT_KIND gadgets be controlled.

  GTNM_FrontPen and GTNM_BackPen to let the color of NUMBER_KIND gadgets be controlled.

  GTNM_Format to specify the formatting string to use with NUMBER_KIND gadgets. This is so a localized number format using "%lD" instead of "%ld" can be used.

  GTNM_MaxFormatLen to specify the maximum length of the string that can be generated by GTNM_Format.

  GTTX_Justification and GTNM_Justification to allow for right and center justification on TEXT_KIND and NUMBER_KIND gadgets.

  GTSL_MaxPixelLen lets you specify the maximum pixel length the level display of a SLIDER_KIND gadget will occupy. This allows proportional fonts to be used with sliders.

  GTSL_Justification specifies how the level display of a SLIDER_KIND gadget is to be justified within the width allocated by GTSL_MaxPixelLen.

  GTLV_MakeVisible for listviews. You pass it an item number and it makes sure it is visible within the listview display.

- Many new tags for GT_SetGadgetAttrsA():

  GTTX_FrontPen, GTTX_BackPen, GTNM_FrontPen, GTNM_BackPen, GTNM_Format GTTX_Justification, GTNM_Justification, GTSL_Justification, and GTLV_MakeVisible all have the same purpose as described for CreateGadgetA() above.

  MX_KIND gadgets now support GA_Disabled.

  GTSL_DispFunc and GTSL_LevelFormat are now changeable via GT_SetGadgetAttrs() instead of being create-time only attributes.

- Added the GTBB_FrameType tag which gives access to the new frame types available in Intuition. You pass the tag to DrawBevelBox() and can specify BBFT_BUTTON, BBFT_RIDGE or BBFT_ICONDROPBOX.

- GT_SetGadgetAttrs() can now safely be called when the gadget being affected is not attached to a window, by passing a NULL window parameter

- Specifying GTTX_CopyText and not GTTX_Text now works for TEXT_KIND gadgets.

- TEXT_KIND or NUMBER_KIND gadgets that have the GTTX_BackPen or GTNM_BackPen tags specified look visually cleaner when changing the gadget text using GTTX_Text or GTNM_Number than those without. Listviews take advantage of this when applicable.

- The value of GTSL_Level is now bounds checked at CreateGadget() time in addition of at GT_SetGadgetAttrs() time.

- Fixed bug where doing GT_SetGadgetAttrs() on a MX_KIND gadget and not passing the GTMX_Active tag would reset the active selection to #0 instead of leaving it alone.

- The NewGadget.ng_TextAttr field can now be NULL whenever a gadget is created. In such a case, the screen's TextAttr is used (screen's TextAttr is determined from the VisualInfo in the NewGadget structure).

- The level display of SLIDER_KIND gadgets is now rendered with background set to BACKGROUNDPEN instead of 0, which is more "correct".

- Now uses SetWriteMask() instead of SetWrMsk().

- Added support for gadget help in all gadget types.

- GadTools now handles the new ExtIntuiMessage generated by Intuition.

- Many enhancements to PALETTE_KIND gadgets:

  Palette gadgets no longer display a box filled with the selected color. The selected color is instead denoted by a box drawn around the color square in the main palette area.

  Palette gadgets now allow strumming, and right mouse button cancellation.

  GTPA_ColorTable is a new tag to support sparce color tables in gadtools. Ths tag can be passed at create/set/get time.

  GTPA_NumColors is a new tag to specify the total number of colors to display. This allows amounts of colors that are not powers of 2. This tag is also good at create/get time.

  GTPA_ColorOffset is now supported at get/set time.

  Renders itself much faster, this makes a big difference on 256 color screens.

  Now does quite smart layout of the color squares. An attempt is made to keep them as square as possible, based on the aspect ratio information obtained from the gfx database. As many colors as possible are put on the screen, until things get too small in which case the upper colors are thrown away.

- MX_KIND gadgets now support ng_GadgetText and will display the label in relation to the group of mx gadgets.

- Added GTMX_TitlePlace tag. This determines where the title of a MX_KIND gadget is displayed. If this tag is not provided, the title is not displayed. This is required for compatibility.

- Fixed size calculation errors in listview present since V37. This may cause certain listviews to change in size from their V37 size.

o Revamp of listviews:

ListView lines can no longer end up complemented in certain tricky situations involving detaching lists.

ListViews correctly track the selected line when you click in them.

ListViews were clipping the text four pixels early on the right.

Added GTLV_CallBack. This tag allows a callback hook to be provided to gadtools for listview handling. Currently, the hook will only be called to render an individual item. This adds the very useful ability to define a callback hook which is used to scroll complex items such as graphics, etc.

Listviews now allow strumming. That is, holding down the left mouse button and moving the mouse up or down causes the active selection to track the mouse. Moving the mouse off the top or bottom of the listview causes the list to scroll.

Listviews that used to have a display or string gadget underneath them now have a highlight bar to indicate the selected item. This is in anticipation of listview multi-selection. If the listview had a display gadget, it no longer does as the highlight bar is used. If a listview had a string gadget, it retains it.

Listviews highlighting is done using the pen-spec method instead of the 1.3 complementing method.

Listviews are much faster at rendering and scrolling, which makes a noticeable difference in 8 bit planes

Added GTLV_MaxPen tag to specify the maximum pen number used by a custom rendering callback hook. This enables more optimal scrolling and screen updates.

(GTLV_Selected, ~0) is now supported at both create and set times.

(GTLV_Labels, ~0) is now supported at create time.

Listviews now support GA_Disabled. This causes the list area to be ghosted, but the scroller and arrows remain unghosted.

o When cloning a rastport for internal use, no longer copies the TmpRas field of the original rastport, which should eliminate some potential bugs.

o Changed the definition of TEXTIDCMP and NUMBERIDCMP in gadtools.h to be (0) instead of (NULL), to keep the Manx compiler happy.

o It is now safe to call GT_GetGadgetAttrs() and GT_SetGadgetAttrs() with NULL gadget pointers.

o When GT_SetGadgetAttrs() is called on an active STRING_KIND or NUMBER_KIND gadget, the gadget is automatically reactivated after its string is changed. Although this reactivation flickers, the functionality is quite useful.

o Now copies a complete TTextAttr structure when needed to fix potential problems with WeightTAMatch(). This is only done to create underlines under gadget labels.

o Fixed example in CreateGadget() autodoc. Only had a single argument in the call to GT_RefreshWindow()

o BOOPSI images are now allowed in gadtools menus.

o Added the GTTX_Clipped tag for TEXT_KIND gadgets.

ramdrive.device

o Uses AllocMem(xxxx,MEMF_REVERSE!MEMF_KICK!MEMF_NO_EXPUNGE) instead of private AllocHigh() code.

o Uses CopyMem() instead of an unrolled loop. CopyMem()'s MOVEM's are faster than the unrolled loop used previously.

o Protects KickTag/KickMem list with FORBID/PERMIT inside of KillRad() vector used by REMRAD. Fixes possible crash if some other task is fiddling with these lists at the same time.

ram-handler

o Fixes the long standing bug where if the file you were examining with ExNext() is deleted, RAM: goes off into never-never land (and your system follows). If the file is deleted, it will restart at the beginning of the directory.

o Disabled softlinks in RAM: to save ROM space.

shell

o NewShell/NewCLI now open full with (like Shell from WB).

o NewShell/NewCLI now handle FROM fields up to 255 long (up from 127) and errors out if FROM or WINDOW are too long.

o Resident module handling now properly Forbid()s around seg_UC++/--.

o Prompt now handles %%.

o Removed two harmless enforcer hits at boottime.

o If a "command `command...` ..." fails, it no longer inserts the error message and continues (FailAt is used to determine failure).

o Added evil kludge to solve the problem of 1.3 SetClock crashing on 68040s.

o Fixes the write to rom on <> redirection.

o NewShell/NewCLI no longer print error messages if no S:Shell-Startup is present.

timer.device

o Timer keys off new GfxBase flag for determining EClock frequency since PAL/NTSC is now software selectable.

trackdisk.device

o Post-write delay has been moved from 2ms (spec is 1.2ms) to 3ms, and side settle delay from 1ms (spec is 0.1ms) to 1.5ms. This should fix most A1010's out there. In addition, both of those values have been made part of the public unit structure like settle delay and step delay, so people with REALLY bad A1010's can back it off as far as they need to (or setpatch can).

o Fixes a nasty oversight in the HD floppy handling. After switching from a HD floppy to LD floppy and back to HD floppy, you could never safely write to an HD floppy unless you formatted an HD floppy first.

What happened was that the extra slop area at the front of the write wasn't getting set to $aaaaaaaa, it was being left with garbage from the last LD read (since LD uses less slop, it's start-read spot is earlier). Format re-inits the entire buffer, as does the first switch to an HD floppy (only the first, since it switches to a larger buffer then).

o Fixed a bug where if a track was totally unreadable it returned the number of retries as the IO_ERROR instead of TDERR_NoSecHdr (this was causing the "Error 11" stuff when you popped a disk while copying from it).

utility.library

- Downcoded all tag calls from C to assembly which yields substantially faster performance.
- Removed 68020-specific versions of the date conversion routines.
- Fixed bug in MapTags() where the "includeMiss" parameter didn't work.
- Cleaned up and expanded autodocs.
- Cleaned up public include files.
- Added comments in the autodoc entries for the 4 32-bit math routines, to the effect that they preserve address registers, and that A6 does NOT have to be loaded in order to call the routines. This is an exception to the standard rule, but can avoid register shuffling which is important in low-level math routines.
- Made the math routines several cycles faster on 68000 machines.
- Added SMult64() and UMult64() which do 32x32=64 bit integer math.
- Added ApplyTagChanges().
- Added two new library calls that are mainly here to help intuition get smaller. These are PackStructureTags() and UnpackStructureTags().
- Added the NameSpace code.
- Added GetUniqueID().

Appendix D: Release 3.01 ROM Changes

Following is a description of most of the important changes made to the ROM-based system software between Release 3 and Release 3.01. This doesn't cover changes made to modules not discussed in this talk. It also doesn't cover any changes after December 1992. Since 3.01 is still under development as of this writing, more changes are likely to be made to the software prior to release.

BootMenu

- o Fixed bug where the chip type mutual exclude gadget was being displayed even on pre-ECS machines.

exec.library

- o Added the full support for the Zorro-III quick interrupts. The new LVO (in an old slot) ObtainQuickVector() is used to allocate the vector. There is no deallocation since this is basically a configuration issue and not a dynamic thing.
- o On machines with PCMCIA cards, EXEC now makes sure the interface is turned on at boot time and then will turn it off before configuration. This should let a full 8-meg of RAM be added in the Zorro-II space. This change requires an update to the credit card resource/device such that it will correctly turn on the interface if needed.
- o The Quick Interrupt vectors that have not yet been added used to be -1. Now they point at an Alert that is the new Unexpected Quick Interrupt.

gadtools.library

- o Fixed bugs in clipping code in TEXT_KIND and NUMBER_KIND gadgets. The clipping didn't work correctly on right and center justified text, and was under-evaluating the number of pixels available for the text in a gadget that didn't have borders.
- o Fixed bug in the calculation of the default value for the GTSL_MaxPixelLen tag. This caused odd clipping of the number display for sliders whenever the title of the gadget wasn't on the same side of the slider as the display of its current value.
- o Fixed GTJ_CENTER option for the various GTXX_Justification tags. The way centering was done could cause certain characters to get lost.

carddisk.device

- o Now flushes cache during data writes in anticipation of 040 copyback cache on A1200 (no hardware support for PCMCIA memory space data cache control provided, so the data cache is still potentially a problem when programming flash rom; means turning off the data cache globally for 030/040 A1200's to support 6-10us write/verify timing).

card.resource

- o Now leaves PCMCIA slot disabled if any RAM is configured at $600000; this allows use of >4Megs of 24bit RAM on the A1200 at the expense of being unable to use the PCMCIA slot.
- o Partial work around for a hardware bug in our PCMCIA implementation which presents 2Meg+ addresses everytime we access ATTRIBUTE memory. This causes a problem when a >2Meg card which ignores REG is used (and a potential problem with any card which tries to decode the entire address when REG is set). The former problem is kludged around by trying to sniff out mirroring of 4 bytes at $A00000 and $800000 but not mirrored at $600000.
  - Tested with:
    - Fujitsu 512K SRAM (ignores REG)
    - Fujitsu 128K SRAM (decodes entire address)
    - Panasonic 512K SRAM (returns $FF for attribute memory)
    - HP 128K SRAM (has 16 bytes of attribute memory)
    - NewMedia 2M PSRAM (ignores REG)
    - NewMedia 4M PSRAM (ignores REG - this is the card which demonstrates the problem)
- o Considerably faster memory sizing for SRAM/DRAM cards (does test of every 256 words/long-words) - tested with:
  - Fujitsu 512K SRAM (ignores REG)
  - Fujitsu 128K SRAM (decodes entire address)
  - Panasonic 512K SRAM (returns $FF for attribute memory)
  - HP 128K SRAM (has 16 bytes of attribute memory)
  - NewMedia 2M PSRAM (ignores REG)
  - NewMedia 4M PSRAM (ignores REG - this is the card which demonstrates the problem)
- o CardMemoryMap structure extended for V39 card.resource. Now has COMMON/ATTR/IO Memory Zone size for lookup via structure. Will be used to provide splitting of memory zones in the future if needed. No change for existing software.
- o BVD1/SC, BVD2/DA, and BSY/IRQ status change interrupts can now be individually enabled/disabled. WR (Write-Protect) status change interrupts are always enabled (rare), and there is no change in the defaults. This is intended for future use if needed (e.g., Flash-ROM which expects software to poll SC during programming; better performance can be obtained if interrupts are not generated). If needed on the A600, this can be implemented as documented work around, or SeFunction() of CardMiscControl(). No expected change for existing software; defaults are the same as they use to be in V37 card.resource. Spurious interrupts (change true, but interrupt disabled) are cleared by the resource software, and hidden from the status change callback hook.
- o Secondary callback option for status change interrupts; allows high-performance hardware to be serviced via interrupts only (instead of signalling a task).
- o Flush Cache when ReleaseCard() is called. A flush before full release ensures that no more writes will occur once the caller returns from ReleaseCard(). This is to support the 040 copyback cache when/if an 040 becomes available for the A1200. Would prefer control over the data cache for PCMCIA space independent of the first 4MEG of 24bit Fast RAM, but we don't have this feature. Lack of Data Cache control for PCMCIA space is still potentially problematic for use of FlashROM programming which requires disabling the DATA cache for 030/040 equipped A1200's so that fast (6-10us) write/read operations can be performed during programming. Disabling the DATA cache during FlashROM writes means disabling globally.

dos.library

- o Fixed bug in RemAssignList(): it wouldn't remove the first lock in the assign.
- o AttemptLockDosList() was returning NULL or 1 for failure instead of NULL.
- o Made RunCommand() free any memory added to the tc_MemEntryList by the command being run. tc_MemEntry is now saved and emptied before calling the command, and restored after any added memory is freed.
- o Fixed ExAll() emulation to not lose 1 file each time the list is broken up into multiple ExAll() calls.
- o Removed broken attempted fix for rda_Buffer. Autodocs now reflect that you must restore rda_Buffer before each call to ReadArgs() if you pass in an RDArgs structure. Now always clears rda_Buffer in FreeArgs().
- o SetVBuf() enabled.

- Changed some prototypes to avoid c++ reserved word "template". Changed VPrintf()/VFPrintf() prototypes to VOID * from LONG * to reduce useless compiler warnings/casts.

- GetDeviceProc() now returns errors better (especially ERROR_NO_MORE_ENTRIES). It used to lose error codes by calling UnLock().

- SetVBuf() now updates the filehandle so it won't overwrite the buffer with a smaller one if SetVBuf() is called before doing buffered IO. Also it doesn't allocate anything if the new size is the same as the old.

- SetVar() now creates subdirectories as needed (including multiple ones) if they do not exist already (in ENV: and in ENVARC: if GVF_SAVE_VAR is set). Also, it now preserves any IoErr() and won't try to save to ENVARC if there is an error saving to ENV:.

- Modified to fix an edge condition which existed when making the mod to SetVBuf().

expansion.library

- New A1200-specific build that can detect CPU Slot RAM ($08000000) if you have a 32-bit addressing CPU installed. The CPU slot area is 128meg in size (just like the A3000) but has the addition of a wrap check at each 1meg of space in the CPU address space to make low-cost RAM expansion possible without jumpers. (It is now possible to get 128Meg SIMMs so a single SIMM on a CPU card could make a 128Meg of FAST RAM system)

  The reason that this has to be A1200 specific (at least for now) is that the behavior of the existing A500/A2000 CPU cards with respect to 32-bit addresses is very undefined. They act very strangely and differently making it very difficult to safely figure out if these cards are operating correctly or not.

filesystem

- Fixed deletion of the destination of a hardlink - this was badly broken in all versions of the FS, DCFS just made it easier to hit. This was causing spurious "Checksum Error on Block 0" errors (and potentially others), especially when UUCP was using a DCFS partition.

- Fixed a return code which would make softlinks not work if a softlink to a directory is in the middle of a path.

- Fixed the buffer overrun on ExAll() with ED_COMMENT if the first character was >$80 (and lost the first character of comments).

- Fixes updating the date of a directory that changes in the parent of that directory's dircache.

- There were old offsetting bugs in the ExAll() filename/comment copying code. When I fixed the code not to copy too many bytes, the clear was being done to the wrong byte.

filesysres.resource

- Now matches the FS version change.

workbench.library

- Adjusted the sizes of the OK/CANCEL and SAVE/CANCEL gadgets in the Workbench requesters to match the rest of the system.

- Fixed a long standing bug that was just found: The system would crash (sometimes) or cause Enforcer hits if files were deleted within a drawer that was also selected for deletion. This one has a fundamental flaw in Workbench which had to be patched with some rather tricky organization of tests...

V39 Specification

ROM and Disk
June 8, 1992
rev 1.6

(©) Copyright 1992 Commodore-Amiga, Inc. All Rights Reserved.
Proprietary and Confidential.

Doc: rev 1.0 Created February 14, 1992

This document contains a description of the modules and programs that have changed for Amiga OS V39. For a low-level function specification please refer to the Autodocs.

Overview

Graphics

- support new AA modes/features
- double buffering
- Retarget sprite and screen/viewport functions
- Palette sharing
- Bitmap functions
- Interleaved BitMaps
- Modepromotion, coercion and selection
- RTG compatible RastPort functions.

Intuition

- Support new AA modes
- Double buffering
- Support AA graphics functions
- Default Busy pointer
- support interleaved bitmaps
- Attached screens (incl. menu lending)
- promotion
- support new sprites
- Gadget help
- Tablet support

- NewLook menus
- Special gadget relativity and GM_LAYOUT method

GadTools
- Palette enhancement
- ListView enhancement
- GT_GetGadgetAttrs() function
- Support for Intuition's NewLook menus and Gadget Help.

- All-new PALETTE_KIND gadget
- Many small enhancements

BootMenu & Battmem
- PAL/NTSC switch
- Chip select switch

Layers
- Use new optimized layers
- Layerinfo backfill hook
- ClipRect traversal

Workbench
- Use ILBM as backdrop pattern
- WB config notification
- Path of file in 'Information'

Console
- fix console not to lock layers for long
- Use Mask bit to speed up rendering.
- Misc. enhancements and fixes.

Clipboard
- fix r/w bug with units above unit 0

Filesystem
- New filesystem with directory blocks.

Exec:
- System specific changes
- Memory pools

- Memory handler
- Procure/Vacate
- Timed out alerts
- SAD

CIA resource

- bump priority to 120

DOS

- Rearrange strings.

Preferences

- New Iprefs
- New WBPattern editor
- New Pointer editor to support AA.
- New Palette editor with pen assignments (using the Colorwheel class).
- Icontrol changes (promotion)

WB Disks

- MultiView
- Localization of selected programs
  - DiskSalv
  - PrepCard
  - MultiView
  - misc. new strings in preferences programs
  - non-concatenated DOS strings.
- DiskSalv. Disk-repair program
- ColorWheel Class: Set of boopsi classes implementing colorwheel colorselection.
- New c: commands to use ASL-file requester and easy-requester from scripts .

Appendix A

New Workbench-disk programs and libraries and their location on the diskset.

Detailed Specifications

The following sections describe in more detail the changes and new features in AmigaOS V39

Graphics Specification:

Philosophy:

The original amiga graphics library exposed many device-dependent details to the application programmer. Because of this, introduction of new graphics devices has been slowed, and application support fo new features has been delayed. To reverse this trend, no features have been added to the new graphics system which cannot be kept for the future. Thus, when newer graphics systems are introduced, system software will need fewer changes, and applications will be ready to automatically use new capabilities. Thus, parameters such as number of bits per pixel, resolution, color palette size, etc are either variable or have been made very large.

Low level functionality

Graphics copper list generation must be extended to handle loading of >6 bit plane pointers, loading of extended color registers, handling of fetch and scrolling for 32 and 64 bit fetch modes, promotion of interlaced modes to 31khz ("flicker-fixing"), promotion of non-interlaced modes via scan-doubling, and color palette banking.

Compatibility

The AA chips are register level compatible with the old and ECS chips at boot up time. However, when new AA modes are enabled and displayed, and a game then takes over the screen display without informing the OS of it, some registers may be in incompatible settings. Two approaches will reduce this:

(a) old Programs which boot with their own custom boot block (games) will have AA features disabled unless they are specifically asked for. This should ensure transparent compatibility for all bootable games.

(b) It will be possible to disable AA features for non-compatible programs. This will be done via the "BootMenu" which is available at system boot time. new options in this menu will allow disabling of AA, disabling of ECS, and switching of PAL and NTSC.

Get/Set functions

This release of graphics.library will attempt to ease the (future) transition to Retargetable Graphics. New functions are provided to do some operations in a more device-independent manner. This will help when we have to support foreign graphics devices, more than 8 bits per pixel, chunky pixels, and true-color displays.

This new calls of "Get/Set" function will allow device independent access to fields in the RastPort structure which were only previosuly manipulable by direct structure access.

Color Map functions

The color palette in AA is different in a lot of ways from the ECS one:

- It has 24 bits per entry, plus one bit to select transparency.
- There are 256 entries which is enough for many programs running on the same screen to share the palette.

The new color palette functions allow for multiple applications to coordinate their access to the palette. This allows applications to, for instance, dynamically remap pictures to match the color palette of the workbench screen, display animations in windows, etc.

All colors are now specified as 32 bit left-justified fractions, and are trunacted based upon the number of bits that the hardware is capable of displaying.

Bitmap functions

These function exist because the new AA chips have alignment restrictions in high bandwidth modes. Changing InitBitMap and AllocRaster to obey these restrictions appears to be very incompatible. Bitmaps created by AllocRaster with a multiple of 32 or 64 pixels per line will be compatible with high fetch modes (2x or 4x respectively). Incompatible ones will fall back to 1x mode, if 1x mode is capable of displaying the screen.

Intuition Specification

New Graphics Modes

Intuition directly supports the new graphics modes.

New Sprite Features

Intuition supports the new sprite-resolutions and sizes. This gives the user a fine-grain pointer, which looks a lot better. As well, Intuition goes to great lengths to remain compatible with users of other sprites.

Intuition now has a default busy pointer.

NewLook Menus and TitleBar

By adding three additional pens to the DrawInfo pen-array, Intuition now allows control over the colors used in rendering menus. This will allow the black on white menus with black trim that the Style Guide recommends. As well, the screen title bar is rendered to match.

In addition, the Amiga-key and checkmark symbols that appear in the menus are colored accordingly, and are scaled by default to the screen's font. It is possible to use a custom font in a window's menus, and have the checkmark and Amiga-key symbols for that window scaled to that font.

It is trivial to write compatible applications that come up optimally under all conditions (including V37).

GadTools recognizes and supports NewLook Menus.

Double-Buffering in Screens

Intuition now supports double-buffering in screens, via the new double-buffering calls in graphics.library. It is possible to have both gadgets and menus (in a borderless backdrop window) on a double-buffered screen.

Attached Screens

Intuition now supports the concept of attached screens. The {SA_Attach,parent} tag is used for this purpose. First open the parent screen. Then open one or more child screens with this tag. Child screens can be dragged independently of each other and their parent (though never above their parent). Pulling down the parent below its natural top causes the child screens to move as well.

Attached screens also depth-arrange as a group, whenever the user does the depth-arranging. Programmatic depth arrangement of the parent does the same. Programmatic depth arrangement of a child moves it alone, but only with respect to the group.

Gadget Help and Special Gadget Relativity

The gadget structure now has an additional rectangle, which is the bounding box. The HelpControl() function can be used to turn on "Gadget Help" for one or more windows. When Gadget Help is on, passing the mouse over the bounding box of a gadget having the GMORE_GADGETHELP property causes an IDCMP_GADGETHELP message to be sent. IDCMP_GADGETHELP messages are also sent when the mouse is not over the window, or is over the window but not over a help-reporting gadget. There is a GM_HELPTEST method which boopsi gadgets can use to refine their sensitivity area or to delegate the test to member gadgets.

Also, GRELxxx gadgets now use the bounding box to erase themselves, so you can now have GREL gadgets whose imagery extends outside of the gadget select box. As well, there is a GM_LAYOUT method which boopsi gadgets can use to do any arbitrary layout within their window.

New Tablet Support

Intuition defines a new subclass of input-event which should solve tablet handling quite nicely. Tablet drivers now have an easy time knowing what they should be scaling to. In addition, it is now possible for applications to receive extended IntuiMessages that can contain such information as pressure, sub-pixel position, etc. The extra information is extensible to cover things we cannot yet forsee.

Custom Image Disable-Rendering

For a disabled gadget under V37, Intuition always rendered the dot pattern itself. Under

V39, an image can set the IA_SupportsDisable attribute to TRUE, and then Intuition will let it do its own disabled rendering, through the IDS_DISABLED or the (new) IDS_SELECTEDDISABLED image drawing states. This is transparently compatible with V37.

Promotion

Crudely speaking, promotion is de-interlacing in software. IControl has a "promotion option". This flags instructs Intuition to tell graphics to change the mapping of the default monitor. Database-aware programs will now receive information about the double-NTSC or double-PAL mode when enquiring about the default monitor. Of course, if the user changes the promotion setting after the application is open, not everything it learned from the database will still be true. The new method also means that an application can request to never be promoted by asking for explicit NTSC or explicit PAL, instead of the default monitor.

Pen-Sharing

Under V39, Graphics supports sharing pens (ObtainPen(), etc.). Intuition now uses and supports the graphics.library pen-sharing scheme. For all screens, pens found in the DrawInfo->dri_Pens are obtained as sharable. For existing public screens, all other pens are allocated as exclusive. For custom screens and aware public screens (those that set the new SA_SharePens tag), all other pens are left unallocated. On the Workbench screen, pens 0-3 are guaranteed to be sharable.

Interleaved Screens

As graphics.library now supports interleaved bitmaps, Intuition makes this feature available to screen openers via the SA_Interleaved tag. Currently, the Workbench is interleaved by default.

Coercion and Inter-Screen-Gap

Intuition now relies on graphics.library to figure out coerced screen modes, and the required inter-screen-gap. This allows greater inter-mode compatibility, better coercion results, and compatibility with future monitors.

Pens and DrawInfo

The new SA_LikeWorkbench tag gives you a screen just like the Workbench. This was basically a consequence of getting pens settable through preferences while keeping NewLook menus on the Workbench screen.

When the SA_Pens pen-array is evaluated, pens are masked to the number of available colors. As well, special definitions of pen-number (PEN_C3, PEN_C2, PEN_C1, and PEN_C0) mean the complementary pen numbers of pens 0-3.

Preferences now listens to only 8 colors for the bitmap, which Intuition will set as the first four and the last four colors of the Workbench or any "Workbench-like" screen (those having the SA_FullPalette or SA_LikeWorkbench attributes).

The way the DrawInfo pens are determined is Intuition picks a default pen-array. Then, any pens you supply with SA_Pens override the defaults, up until the ~0 in your array. If the screen is monochrome or old-look, the default will be the standard two-color pens. If the screen is two or more planes deep, the default will be the standard four-color pens, which now include the new-look menu colors.

If the screen has the SA_LikeWorkbench property, the default will be the user's preferred pen-array, changeable through preferences. There is a preferred pen-array for four colors, and one for eight or more colors.

Miscellaneous Window Features

If a window is not frontmost, but nevertheless is unobscured, then clicking on its depth gadget used to bring it to front, which had no visual effect. It now depth-arranges correctly (i.e. to back).

It is now possible to receive an IDCMP_CHANGEWINDOW message when a window is depth-arranged.

Size-only zooming (where the window top-left corner is preserved) is now supported.

Miscellaneous Screen Features

Screens may now be made non-draggable. Screens can be designated as exclusive, meaning they will never share the display with another screen. The SA_BackFill tag can be used to specify a backfill hook for the screen's LayerInfo. Is it now possible to

control the SA_Pens or other previously unavailable attributes of the Workbench screen. This will allow things such as light-on-dark text. Screens can have interleaved bitmaps. There are new universal screen movement and depth-arrangement functions. Intuition now supports scrollable 2024/Moniterm screens.

Screen depth is always validated. Making a request for a screen too deep causes failure with OSERR_TOODEEP.

Miscellaneous

The DoGadgetMethodA() function allows you to invoke a method on a gadget, which will then receive a valid GadgetInfo pointer.

The boopsi image class "frameiclass" supports the standard border types (eg. bevelled box, string-gadget ridge, "moat", AppWindow icon drop-box).

Intuition now notices and repairs damage when boopsi gadgets use ScrollRaster(). (Such damage occurs when the gadget is in a simple-refresh window and part of the scrolled area is obscured). Such gadgets MUST set GMORE_SCROLLRASTER in order to benefit from this magic repair feature. (Note that ScrollWindowRaster() is for applications. Boopsi gadgets ought to use ScrollRaster() or ScrollRasterBF()).

Speed Improvements

When a window opens active, its border is rendered only once, not twice. This looks better and is faster.

EasyRequests and AutoRequests now put their imagery and gadgets directly in their window, instead of also using a requester. This saves memory, and is faster.

The rubber-band frame when moving or dragging windows is now drawn faster.

Several other speed optimizations were taken.

Gadtools specification

For V39, work has been done to GadTools to address new Intuition features, to add

some important features missing from V37 GadTools, and to fix a few bugs. The key changes are:

- A "get gadget attributes" function

- Support for Intuition's NewLook menus

- Support for Intuition's new Gadget Help feature

- Many enhancements to the different kinds of gadget, including a complete re-write of PALETTE_KIND, support for proportional fonts in the slider level display, as well as scalable checkmark and radio button imagery.

BootMenu & Battmem:

Bootmenu has been enhanced with support for selection of default video mode (PAL or NTSC) and selection of default chipset.

Bootmenu will switch between PAL and NTSC when a key on the keyboard is pressed (except certain special purpose keys like SHIFT, ALT etc).

Chipselection offers the following choices:

- Original [the original pre ECS chipset]
- ECS
- Best

'Best' means that the system will boot into the 'AA' on a AA system, ECS on an ECS system etc.

Setting can be saved to battmem or used without saving. Support for saving to battmem may be missing from the first releases.

Layers specification:

The new optimized layers.library is used in V39. Layers has been further enhanced by adding ClipRect traversal and support for Layerinfo backfill hook

Workbench specification:

Workbench has been updated to support ILBM backdrop patterns. As a further enhancement 'Information' is now asynchronous and shows the complete filename path.

Console specification

- AA scrolling optimizations
- AA change in EOR cursor logic
- Fix CMD_CLEAR
- Research a command to specify new defaults for ESC[0m. This would allow programs which send ESC[0m not to change the users preferred defaults - a nice thing for many of our programs which run from the Shell, and insist on reseting my preferred SGR settings.
- Cursor & highlighting only use colors 0-3 in screens which support color sharing (e.g., WB).

Clipboard.device specification:

Fix units 1-255 multi R/W handling.

Filesystem specification

A new filesystem type (option) has been added: Directory Caching. This caches all the information normally returned by ExNext/ExAll in a series of blocks attached to each directory. These blocks normally hold 15 entries; the actual number can range from 3 to 20 for 512-byte blocks.

These blocks allow exnext/exall to do drastically less disk access. Roughly, this makes floppies almost as fast as harddisks under 2.04, and harddisks almost as fast as the ramdisk. This is dramatically noticable to users, especially floppy-based users.

There is a slowdown in file-creation (about 30% on harddisks), some slowdown on delete and on close of a modified file due to the need to update these caches. Also, validation, if needed, may take slightly longer since it must validate all the directory

caches. These caches do add to the ability of disk-repair utilities to rebuild the directory structure of a trashed disk, though, or rebuild trashed fileheader blocks.

The code cost is about 2-2.5K. Also, in the process of doing these modifications, several nasty holes from 2.04 (and before) were fixed. These include the multiple-files-of-the-same-name bug, and a bug not seen experimentally where a rename of a file and a delete of the previous file in that hash chain could collide, losing entire strings of files (or in rename versus rename, entire strings of files could be moved to the wrong directory). All modification of directory structures is now locked behind coroutine semaphores, so no holes like this should show up again.

Various other bugfixes are being done as required.

CIA reource specification:

Bump priority to 120

DOS specification:

Rearrange strings to avoid using concatenation.

Preferences specification:

Prefs/Pointer

Changes are required for AA sprite support. This includes larger sprites with varying resolutions and number of colors. Another needed change is to support editing of the default busy pointer.

Prefs/Palette

With 3.0's pen sharing abilities, a different more abstract color model must be available to users. Palette prefs will gain a pen editing ability which will let the user associate arbitrary RGB values to any of Intuition's logical pens. So for example, the user will be able to specify which RGB value should be used for his text, background, window border, etc...

Prefs/WBPattern

Must learn to deal with more complex patterns. This will be done in a manner similar to the current Sound preferences in that a user can just enter the name of a picture to display as Workbench pattern.

Prefs/IControl

A few checkboxes need to be added to enable control of Promotion.

C/IPrefs

Needs to adapt to the new extended Workbench pattern ability. It must load an ILBM file and pass the result on to Workbench. Operations can be done on the graphics before displaying it such as color remapping.

Needs to adapt to the new way Workbench expects to hear about changes to Font and Locale preferences.

WB disk (tools, utilities etc.) specification

Tools/DiskSalv

New program to repair and recover hard drives.

Utilities/MultiView

This is a object-oriented data viewer that uses the datatypes.library for its object handling. Any type of data file can be viewed using this viewer, as long as class for the data type is available.

Objects can be viewed on the Workbench screen, any public screen, or a custom screen (for HAM pictures for example).

Classes that will be shipped will be:

ANSI - Used to view any text document that Utilities/More could have viewed, including decoding of ANSI escape sequences.

ILBM - Used to view Amiga ILBM pictures. Includes the ability to remap a picture to the current palette (uses new graphic functions to obtain pens, plus does simple dithering).

AmigaGuide - Used to browse through AmigaGuide documents. This is backwards compatible with the existing AmigaGuide. There have been a number of small enhancements, such as; proportional font support, word wrap, and text attributes (bold, italic, and underline).

Storage/Monitors

New monitors for AA support

Localization

Localize the following applications:

- DiskSalv
- PrepCard
- MultiView
- misc new strings in updated preferences programs.
- new non-concatenated DOS strings.

Tools/Commodities/MouseBlanker

A new mouse blanking commodity that shuts off the mouse pointer whenever the user types on the keyboard.

Tools/Commodities/NumericPad

A new commodity emulating numeric keypad. Mainly intended for use on A600 class systems without numeric keypad.

Appendix A

Workbench 3.0 Disk-changes (layout)

program	Change
C/#?	- Misc. bug fixes
C/AddDataTypes	- New
C/IPrefs	- pointer, palette, and wbpattern changes
C/RequestChoice	- New
C/RequestFile	- New
Classes/amigaguide.class	- New
Classes/ascii.class	- New
Classes/colorwhell.class	- New
Classes/floatstring.class	- New
Classes/ilbm.class	- New
Classes/picture.class	- New
Classes/text.class	- New
Devs/DataTypes/AmigaGuide	- New
Devs/DataTypes/FTXT	- New
Devs/DataTypes/ILBM	- New
Devs/printer.device	- HAM-8 support
L/FastFileSystem	- DCFS version
Libs/amigaguide.library	- New
Libs/datatypes.library	- New
Libs/iffparse.library	- Optimization
Locale/Catalogs/#?	- Updated translations
Prefs/Palette	- Color whell
Prefs/Pointer	- AA support
Prefs/WBPattern	- WB backdrop support
Storage/Monitors/A2024	- 3.0 ROM support
Storage/Monitors/DoubleNTSC	- New
Storage/Monitors/DoublePAL	- New
System/Format	- DCFS support
Tools/Commodities/MouseBlanker	- New
Tools/Commodities/NumericPad	- New
Tools/DiskSalv	- New
Tools/HDToolBox	- UI overhaul

Utilities/More	- Make it look like MultiView from the WB
Utilities/MultiView	- New

This was brought to you

from the archives of

http://retro-commodore.eu